BAB 1
PENGENALAN C++

1.1 Pendahuluan

Pada mulanya bahasa komputer digunakan untuk membantu dalam melakukan perhitungan-perhitungan telemetri. Pada saat itu, bahasa yang digunakan masih primitif, yakni bahasa mesin yang hanya mengenal angka 1 dan 0. Selanjutnya bahasa mesin tersebut disederhanakan menjadi bahasa-bahasa yang bisa dipahami dengan istilah *mnemonic* seperti ADD,MOV,JMP dan lainnya. Bahasa tersebut kita kenal dengan bahasa *assembly* (bahasa tingkat rendah).

Pada tahun 1969, laboratorium Bell AT&T di Murrat Hill, New Jeresey menggunakan bahsa *assembly* ini untuk mengembangkan siste operasi UNIX. Maksudnya adalah untuk membuat sistem operasi yang dapat bersifat *'programmer-friendly'*. Setelah UNIX berjalan, Ken Thompson, seorang pengembang sistem di laboratorium tersebut mengembangkan bahasa baru dengan nama bahasa B. Huruf B diambil dari BCPL (Basic Combined Programming Languange). Bahasa ini kemudian digunakan untuk menulis ulang atau merevisi sistem operasi UNIX. Oleh karena bahasa B ini masih bersifat interpret dan lambat, maka pada tahun 1971, sistem operasi UNIX kemudian ditulis ulang dengan menggunakan bahasa C, yaitu bahasa pemrograman yang dikembangkan oleh Dennis Ritchie, seorang pengembang di laboratorium yang sama.

Sampai saat ini, bahasa C masih digunakan untuk melakukan pengembangan-pengembangan program dan sistem-sistem operasi, diantaranya sistem operasi Windows. Alasan itulah yang menjadikan bahasa C sangat populer di dunia pemrograman, khususnya di industri perangkat lunak. Kelemahan dari bahasa C adalah masing tergolong susah untuk dipelajari karena masih bersifat prosedural murni. Untuk membentuk satu objek, kita harus melakukan penulisan kode yang banyak. Untuk mengatasi masalah tersebut, pada tahun 1983, seorang doktor bernama Bjarne Stroustrup menciptakan bahasa baru yaitu bahasa C++ yang merupakan bahasa *hybrid* dari bahasa C. Bahasa C++ didasarkan atas bahasa

C sehingga kita dapat melakukan kompilasi program-program yang ditulis dalam bahasa C dengan menggunakan kompiler C++.

Pada mulanya C++ disebut "a better C". Nama C++ sendiri diberikan oleh Rick Mascitti pada musim panas 1983. Adapun tanda ++ berasal dari nama operator penaikan pada bahasa C. Keistimewaan dari bahasa C++ adalah dapat mendukung pemrograman berorientasi objek atau dikenal dengan istilah Object Oriented Programming (OOP).

1.2 C dan C++

Bahasa C dan C++ digolongkan ke dalam bahasa tingkat menengah (*middle level languange*). Seorang profesor yang bernama Niklaus Wirth di Politeknik Zurich (Swiss) mengembangkan bahasa tingkat tinggi (*high level language*) yang disebut bahasa Pascal untuk mengajarkan kepada mahasiswanya.

Sebagai bahan pertimbangan, berikut ini pengelompokan tingkatan dari bahasa pemrograman.

Bahasa Tingkat Tinggi	Ada
	Modula-2
	Pascal
	COBOL
	FORTRAN
	BASIC
Bahasa Tingkat Menengah	Java
	C++
	C
	FORTH
Bahasa Tingkat Rendah	Macro-Assembler
	Assembler

Dari tabel tersebut dapat kita lihat bahwa bahasa pemrograman yang terdapat pada bagian atas merupakan bahasa pemrograman yang paling mudah untuk dipahami. Sebagai contoh, bahasa C adalah bahasa yang lebih sulit

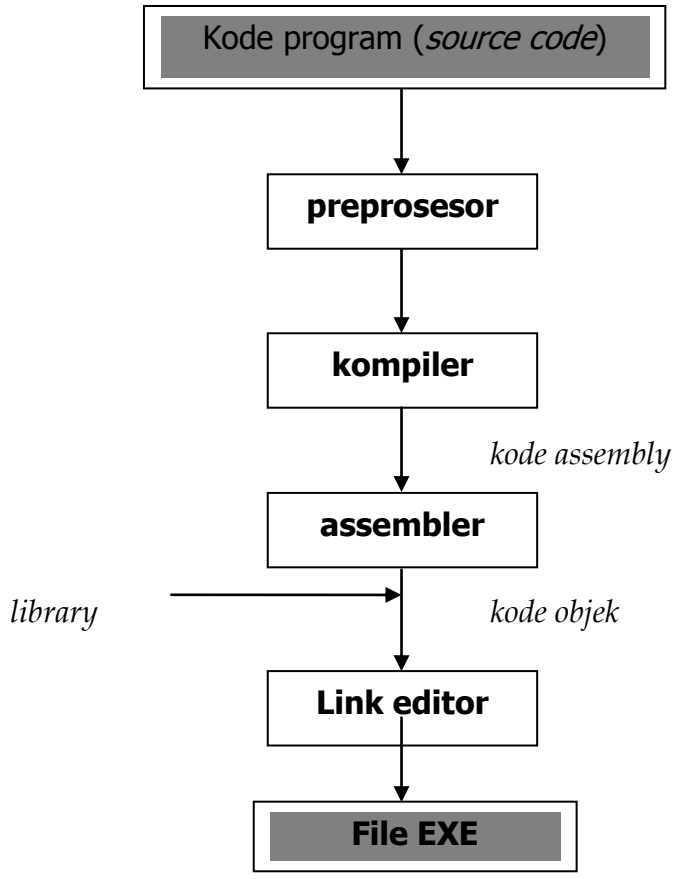
dibandingkan C++ dan C++ adalah l;ebih sulit dibandingkan dengan bahasa Java, dan seterusnya.

Menurut Bjarne Stroustrup (pencipta C++), alasan mengapa C diambil sebagai bahasa dasar dari pembentukan bahasa C++ adalah sebagai berikut:

1. Dapat dihubungkan dengan bahasa tingkat rendah
2. Berjalan dimanapun dan untuk masalah apapun
3. Berjalan mulus dalam sistem operasi UNIX

1.3 Konsep Kompilasi dan Eksekusi Program

Sebelum melangkah lebih jauh dengan pembentukan program C++, maka sebaiknya kita mengetahui terlebih dahulu konsep dari sebuah kompilasi dan eksekusi program di dalam C/C++. Berikut ini gambar yang mengilustrasikan proses kompilasi dan eksekusi program di dalam C/C++.



Gambar Ilustrasi proses kompilasi dan eksekusi program di dalam C++

1.3.1 Preproccesor

Langkah awal adalah memasukkan kode ke dalam bagian preprosesor, yaitu diawali dengan tanda # (*pound*) dan menghasilkan file yang akan dilewatkan ke dalam kompiler. Beberapa preprosesor tersebut adalah

- #include
- #define
- #ifdef
- Dan lain-lain

1.3.2 Kompiler C++

Kompiler akan menerjemahkan kode program yang telah dilewatkan oleh preprosesor ke dalam bahasa *assembly*.

1.3.3 Assembler

Assembler menerima keluaran dari kompiler C++ dan akan membuat sebuah kode objek. Jika dalam kode program kita tidak menggunakan fungsi-fungsi yang terdapat pada library lain, maka kode objek ini akan langsung dieksekusi menjadi file EXE.

1.3.4 Link Editor

Bagian ini dikerjakan jika kode program yang kita buat menggunakan fungsi-fungsi luar yang disimpan dalam suatu library lain. *Link Editor* akan mengkombinasikan kode objek dan library yang ada untuk menjadikan sebuah file EXE.

1.4 Program C++

Program C++ dapat dibuat menggunakan sebarang editor teks maupun editor sekaligus compilernya. Program utama berekstensi (.CPP). Pada saat kompilasi program utama bersama dengan file header (.h) akan diterjemahkan oleh compiler menjadi file obyek (.OBJ). Selanjutnya file obyek ini bersama-sama

dengan file obyek lain dan file library (.LIB) dikaitkan menjadi satu oleh linker. Hasilnya adalah file (.EXE) executable.

1.5 Compiler C++

Compiler C++ yang telah beredar di pasaran antara lain Microsoft C/C++ dan Visual C++. Keduanya dari Microsoft. Sementara Borland international juga mengeluarkan Turbo C++ dan Borland C++.

1.6 Object Oriented Programming (OOP)

Ide dasar OOP adalah mengkombinasikan data dan fungsi untuk mengakses data menjadi sebuah kesatuan unit. Unit ini dikenal dengan obyek. Sebagai gambaran untuk mempurmudah memahaminya, obyek sebenarnya dapat mencerminkan pola kerja manusia sehari-hari. Sebuah obyek dapat diibaratkan sebagai departemen di dalam sebuah perusahaan bisnis, misalnya departemen

- penjualan
- akunting
- personalia

Pembagian departemen dalam perusahaan merupakan upaya untuk memudahkan pengoperasian perusahaan. Sebagai gambaran, jika Anda seorang manajer penjualan di kantor pusat ingin mengetahui data para salesmen di kantor cabang, apa yang Anda lakukan? Langkah yang Anda tempuh pasti bukan datang ke kantor cabang dan mencari data-data tersebut. Untuk memudahkan tugas Anda cukup Anda menyuruh sekretaris untuk meminta informasi. Masalah bagaimana dan siapa yang mencarikan bukanlah urusan Anda. Analogi dengan hal itu, kalau seseorang bermaksud menggunakan obyek, ia cukup mengirim pesan ke obyek dan obyek itu sendiri yang akan menanganinya.

1.7 C++ Klasik dan C++ Modern

Ada perbedaan antara kompiler C++ lama (klasik) dan C++ modern (C++ yang telah memenuhi standar ANSI/ISO). Pada C++ lama masih menggunakan

namespace global, sedangkan di C++ modern menggunakan namespace **std.** Hal inilah yang menyebabkan terdapatnya perbedaan penulisan antara C++ lama dan standar.

1.7.1 Pada Kompiler C++ Lama

```
#include <iostream.h>
int main()
{
        .............
        return 0;
}
```

1.7.2 Pada Kompiler C++ Standar

```
#include <iostream>
Using namespace std;
int main()
{
        .............
        return 0;
}
```

Seperti yang kita lihat bahwa pada C++ standar penulisan file *header* **iostream** sudah tidak diikuti lagi dengan ekstensi **.h**. Untuk menuliskan kode kedua, diperlukan kompiler C++ yang telah mendukung semua fitur yang terdapat dalam C++ standar atau dilengkapi dengan IDE (Integrated Development Environment).

1.8 Fungsi main()

Program C++ memang tidak akan pernah lepas dari suatu fungsi/function. Hal ini karena merupakan ciri OOP. Sebuah program C++ minimal memiliki satu fungsi yaitu main(). Fungsi ini merupakan awal program utama. Tulisan main() merupakan nama fungsi, sedangkan bagian yang diapit dengan { dan } disebut

blok (tubuh fungsi). Dalam hal ini { merupakan tanda awal blok dan } adalah tanda akhir blok. Seperti halnya dalam Pascal, { dalam Pascal identik dengan BEGIN, sedangkan } identik dengan END. Perintah void bermakna bahwa fungsi main() tidak mengembalikan nilai/value.

BAB 2
KOMENTAR, IDENTIFIER DAN TIPE DATA

2.1 Pendahuluan

Pada setiap pemrograman, kita tidak dapat terlepas dari penggunaan tipe data dan pengenal atau pengidentifikasi (*identifier*). Sebagai contoh untuk program yang menampilkan sebuah teks dan bialngan. Misalnya seperti program dibawah ini.

```
#include <iostream>
using namespace std;
int main()
{
        cout<<"Contoh membuat teks"<<endl;
        cout<<2+7;
        return 0;
}
```

Statemen cout di atas berfungsi untuk menampilkan keluaran (*output*) ke layar monitor. Statemen ini hanya akan dikenali jika kita mendaftarkan library iostream yang tersimpan dalam namespace std. Fungsi endl berfungsi untuk memindahkan cursor ke baris baru.

2.2 Komentar

Komentar program didefinisikan sebagai bagian dari sintaks program yang tidak ikut dibaca pada saat proses kompilasi. Dalam bahasa C++, komentar dibagi menjadi dua jenis yaitu komentar yang hanya terdiri dari satu baris dan komentar yang terdiri dari beberapa baris. Berikut ini penjelasan dari keduanya.

2.2.1 Menggunakan Tanda //

Tanda ini digunakan untuk menuliskan komentar yang banyaknya hanya satu baris. Jika kita menggunakan tanda ini untuk menuliskan komentar yang

lebih dari satu baris, maka teks di belakang tanda // tetap menjadi komentar. Contoh penggunaan tanda ini adalah sebagai berikut.

```
// Ini adalah komentar untuk satu baris
```

2.2.2 Menggunakan Tanda /*...*/

Berbeda dengan tanda //, tanda ini dapat digunakan untuk menuliskan komentar yang banyaknya satu baris atau lebih. Komentar ini dimulai dari tanda /* sampai ditemukan tanda */. Contoh penggunaanya dapat dilihat dibawah ini.

```
/* Ini adalah komentar yang banyaknya satu baris */
/* Ini adalah komentar panjang
        Yang banyaknya
        Lebih dari satu baris */
```

2.3 Identifier

Identifier merupakan suatu pengenal atau pengidentifikasi yang kita deklarasikan agar kompiler dapat mengenalinya. Identifier sendiri dapat berupa variabel, konstanta, fungsi, kelas, template maupun namespace.

Identifier yang berfungsi sebagai variabel dan konstanta berfungsi untuk menampung sebuah nilai yang digunakan dalam program. Identifikasi ini dilakukan untuk mempermudah proses penanganan data atau nilai, misalnya untuk memasukkan dan menampilkan nilai.

```
#include <iostream>
Using namespace std;

Int main()
{
        Char Teks[100];
        int X;
        cout<<"Masukkan sebuah kata    : "; cin>>Teks;
        cout<<"Masukkan sebuah angka : ";      cin>>X;
        cout<<Teks<<endl; //bisa ditulis dengan cout<<X<<'\n';
        cout<<X;

return 0;
}
```

Hal-hal yang perlu diperhatikan dalam membuat identifier antara lain sebagai berikut:

1. Bersifat case sensitive, maka dalam C++ akan membedakan variabel yang ditulis dengan huruf besar dan huruf kecil
2. Identifier tidak boleh berupa angka atau diawali dengan karakter yang berupa angka. Misal

   ```
   long 1000;// SALAH karena identifier berupa angka
   long 2X;          // SALAH karena identifier diawali oleh karakter angka
   long X2           // BENAR karena identifier tidak diawali oleh angka
   ```
3. Identifier tidak boleh mengandung spasi
4. Identifier tidak boleh menggunakan karakter-karakter simbol (#,@,?,!,$, dll)
5. Identifier tidak boleh menggunakan kata kunci yang terdapat pada C++.(Misal break,return dll)
6. Nama identifier sebaiknya disesuaikan dengan kebutuhannya
7. Hindarilah penggunanaan nama identifier yang sama dengan identifier yang digunakan oleh C++.

2.3.1 Konstanta

Konstanta adalah jenis identifier yang bersifat konstan atau tetap, artinya nilai dari konstanta di dalam program tidak dapat diubah. Untuk pendeklerasian konstanta adalah sebagai berikut:

```
const tipe_data nama_konstanta = value;
```

2.3.2 Variabel

Variabel adalah identifier yang mempunyai nilai dinamis, artinya bahwa nilai variabel tersebut dapat kita sesuaikan dengan kebutuhan kita dalam program.

```
tipe_data nama_variabel; atau
tipe_data nama_variabel1, nama_variabel2, nama variabel3;
```

2.4 Tipe Data

Tipe data berfungsi untuk mempresentasikan jenis dari sebuah nilai yang terdapat dalam program. Dalam bahasa C++, tipe data dibagi menjadi tiga bagian dasar yaitu tipe dasar, tipe bentukan dan tipe enumerasi.

2.4.1 Tipe Data Dasar

Tipe dasar ini digolongkan ke dalam tipe bilangan bulat (integer), bilangan riil (floating-point), tipe logika (boolean), tipe karakter/teks (character/string).

a. Tipe Bilangan Bulat

Tipe ini digunakan untuk data-data angka yang tidak mengandung angka dibelakang koma. Tipe data yang termasuk ke dalam kategori ini adalah seperti yang terlihat pada tabel dibawah ini.

Tipe Data	Ukuran (dalam bit)	Rentang
Int	16 atau 32	-32.768 sampai 32.767 atau -2,147,483,648 sampai 2,147,483,647
unsigned int	16 atau 32	0 sampai 65.535 atau 0 sampai 4,294,967,295
signed int	16 atau 32	Sama seperti **int**
short int	16	-32.768 sampai 32.767
unsigned short int	16	0 sampai 65.535
signed short int	16	Sama seperti **short in**
long int	32	-2.147.483.648 sampai 2.147.483.648
signed long int	32	Sama seperti **long int**
unsigned long int	32	0 sampai 4,294,967,295

Contoh program yang menggunakan tipe bilangan bulat adalah sebagai berikut:

```
#include <iostream>

using namespace std;

int main() {
  int x; // Mendeklarasikan variabel x dengan tipe data int
  x = 3; // Melakukan assigment terhadap variabel x
  cout<<"Nilai x = "<<x;

  return 0;
}
```

Hasil yang diperoleh dari program diatas adalah sebagai berikut: Nilai x = 3

b. Tipe Bilangan Riil

Tipe ini adalah tipe yang merepresentasikan data-data bilangan yang mengandung angka di belakang koma. Adapun tipe data yang termasuk dalam kategori ini adalah seperti yang ditunjukkan dalam tabel berikut:

Tipe Data	Ukuran (dlm bit)	Rentang	Presisi
Float	32	1.2E-38 sampai 3.4E+38	6 digit presisi
Double	64	2.3E-308 sampai 1.7E+308	15 digit presisi
long double	80	3.4E-4932 to 1.1E+4932	19 digit presisi

Contoh program yang menggunakan tipe bilangan bulat adalah sebagai berikut:

```
#include <iostream>
using namespace std;

int main() {
  double y; // Mendeklarasikan variabel y dengan tipe
          // data double
  y = 222.134; // Melakukan assigment terhadap variabel x
  cout<<"Nilai y = "<<x;
  return 0;
}
```

Hasil yang diperoleh dari program di atas adalah: Nilai y = 222.134

c. Tipe Logika

Tipe ini merepresentasikan data-data yang hanya mengandung dua buah nilai, yaitu nilai logika (boolean). Nilai logika itu sendiri hanya terdiri dari nilai benar (1) dan salah (0). Dalam bahasa pemrograman, nilai ini umumnya lebih dikenal dengan nilai true (benar) dan false (salah).

d. Tipe String

Tipe ini merepresentasikan data-data yang berupa karakter. Tipe ini dinyatakan dengan tipe char, sedangkan untuk string (=kumpulan karakter) dinyatakan sebagai pointer dari tipe char , yaitu ditulis dengan char*. Dalam C++, tipe karakter diapit oleh tanda petik tunggal (‘), sedangkan tipe string diapit oleh tanda petik ganda (“). Adapun tipe data dalam kategori tipe ini adalah seperti berikut:

Tipe Data	Ukuran (dlm bit)	Rentang
char	8	-128 sampai 127 atau 0 sampai 255
unsigned char	8	0 sampai 255
signed char	8	-128 sampai 127

Berikuti ini adalah program yang menunjukkan penggunaan tipe data char dan char*.

```
#include <iostream>
using namespace std;

int main() {
 // Mendeklarasikan variabel Karakter dengan nilai 'A'
 char Karakter = 'Teknik Elektro';
 // Mendeklarasikan variabel Teks dengan nilai "Kata"
 char* Teks = "FT";
 char TEKS[10] = "UNY";
 cout<<Karakter<<endl;
 cout<<Teks<<endl;
 cout<<TEKS<<endl;
 return 0;
}
```

Hasil yang diperoleh dari program diatas adalah sebagai berikut:

```
Teknik Elektro
FT
UNY
```

2.4.2 Tipe Data Bentukan

Tipe data bentukan yaitu tipe data yang dibuat sendiri sesuai kebutuhan dalam program yang akan kita buat. Dalam bahasa latin sering disebut *user defined types.* Adapun yang termasuk dalam kategori tipe bentukan ini adalah array (larik), struktur dan enumerasi.

a. Struktur

Yaitu tipe data bentukan yang menyimpan lebih dari satu variabel bertipe sama maupun berbeda. Berikut bentuk umum pendeklerasian tipe data struktur dalam bahasa C++.

```
Struct nama_struktur {
 tipe_data variabel1;
 tipr_data variabel2;
 .....
};
```

b. Enumerasi

Yaitu tipe data yang nilainya terbatas pada nilai-nilai yang telah didefinisikan saja. Tipe ini digunakan untuk membentuk tipe data yang nilainya bersifat pasti. Berikut bentuk umum pendeklerasian tipe data enumerasi dalam bahasa C++

```
enum nama_tipe { nilai_1, nilai_2,....}
```

sebagai contoh, dibawah ini dituliskan contoh-contoh pendefinisian tipe enumerasi

```
enum HARI { Minggu, Senin, Selasa, Rabu, Kamis, Jum'at, Sabtu};
enum JENIS_KELAMIN { Pria, wanita};
```

Contoh program struktur

```
#include <iostream>
using namespace std;
int main() {
  struct MAHASISWA {
    char NIM[11];
    char Nama[25];
    char Alamat[20];
    char Kota[15];
  };
          MAHASISWA A;   // Mendeklarasikan variabel A yang
              // bertipe MAHASISWA

  A.NIM = "07501241026";
  A.Nama = "Hasbu";
  A.Alamat = "Demangan";
  A.Kota = "Yogyakarta";

  // Menampilkan nilai yang diisikan ke layar
  cout<<A.NIM<<endl;
  cout<<A.Nama<<endl;
  cout<<A.Alamat<<endl;
  cout<<A.Kota<<endl;

  return 0;
}
```

Maka hasil program struktur diatas adalah sebagai berikut:

```
07501241026
Hasbu
Demangan
Yogyakarta
```

2.5 Null Terminated String

Dalam C++, terdapat beberapa fungsi siap pakai yang berguna untuk menyelesaikan masalah-masalah yang berkaitan dengan string. Untuk menggunakan fungsi-fungsi tersebut kita harus mendaftarkan ke file header **string.h**. Berikut ini fungsi-fungsi yang dimaksudkan:

a. Fungsi **strcpy()**

Bentuk umum dari fungsi strcpy() adalah sebagai berikut

char strcpy (char S1, const char* S2);*

Fungsi ini berguna untuk melakukan penyalinan (copy) string dari S2 ke S1.

Contoh program

```
#include <iostream>
#include <cstring>
using namespace std;
int main() {
 char S1[50];
 char* S2;
 // Melakukan pengisian nilai terhadap variabel S2
        S2 = "Ini adalah contoh penyalinan string";
 // Melakukan penyalinan string dari variabel S2 ke
 // variabel S1
 strcpy(S1, S2);
 // Menampilkan nilai dari variabel S1
 cout<<S1<<endl;
 return 0;
}
```

Hasil program diatas adalah

```
Ini adalah contoh penyalinan string
```

b. Fungsi **strncpy()**

Bentuk umum dari fungsi strcpy() adalah sebagai berikut

char strncpy (char S1, const char* S2, size_t_n);*

Fungsi ini berguna untuk menyalin string sebanyak *n* karakter dari variabel S2 ke variabel S1. Contoh penggunaannya adalah sebagai berikut:

```
#include <iostream>
#include <cstring>

using namespace std;
int main() {
  char S1[50];
  char* S2;
// Melakukan pengisian nilai terhadap variabel S1 dan S2
  S1 = "CONTOH";
  S2 = "salin string";

// Melakukan penyalinan string dari variabel S2 ke  variabel S1
  strcpy(S1, S2, 4);

  // Menampilkan nilai dari variabel S1
  cout<<S1<<endl;

  return 0;
}
```

Hasil yang diperoleh dari program di atas adalah sebagai berikut:

```
saliOH
```

c. Fungsi **strdup()**

Bentuk umum dari fungsi strdup() adalah sebagai berikut

```
char* strdup (const char* S);
```

Fungsi ini mirip dengan fungsi strcpy (), yaitu untuk melakukan duplikasi string. Contoh penggunaannya dalam program adalah sebagai berikut :

```
#include <iostream>
#include <cstring>
using namespace std;
int main() {
  char* S1;char* S2;
  // Melakukan pengisian nilai terhadap variabel S1
  S1 = "Contoh Duplikasi String";
  // Melakukan duplikasi string dari varaibel S1 ke variabel S2
  S2 = strdup(S1);
  // Menampilkan nilai dari variabel S2
  cout<<S2<<endl;
return 0;
}
```

Hasil yang diperoleh dari program diatas adalah

```
Contoh Duplikasi String
```

d. Fungsi **strcat()**

Bentuk umum dari fungsi strcat() adalah sebagai berikut

```
char* strcat (char*S1, const char* S2);
```

Fungsi ini menggabungkan (*concat*) string dari S2 ke posisi akhir dari S1.

Contoh penggunaan program adalah sebagai berikut:

```
#include <iostream>
#include <string>

using namespace std

int main() {

 char* S1;
 char* S2;
 char* spasi = " ";

 // Melakukan pengisian nilai terhadap variabel S1 dan S2
 S1 = "Teks";
 S2 = "Gabungan";

 // Menambahkan karakter spasi (' ') ke dalam variabel S1
 strcat(S1, spasi);
 // Melakukan penggabungan string yang terdapat
 // pada variabel S1 dan S2
 strcat(S1, S2);

 // Menampilkan nilai dari variabel S1
 // setelah proses penggabungan
 cout<<S1<<endl;

 return 0;
}
```

Hasil program adalah sebagai berikut

```
Teks Gabungan
```

e. Fungsi **strncat()**

Bentuk umum dari fungsi strncat() adalah sebagai berikut

```
char* strncat (char*S1, const char* S2, size_t n);
```

Fungsi ini berguna untuk menggabungkan *n* karakter dari variabel S2 ke posisi akhir dari variabel S1. Contoh penggunaan program adalah sebagai berikut

```
#include <iostream>
#include <cstring>

using namespace std;

int main() {
  char* S1;
  char* S2;

  // Melakukan pengisian nilai terhadap variabel S1 dan S2
  S1 = "Nama saya adalah ";
  S2 = "Budi Raharjo";

  // Menggabungkan 4 karakter dari variabel S2 ke variabel S1
  strncat(S1, S2, 4);

  // Menampilkan nilai dari variabel S1
  // setelah dilakukan penggabungan
  cout<<S1<<endl;

  return 0;
}
```

Hasil program diatas adalah sebagai berikut

```
Nama saya adalah Budi
```

f. Fungsi **strlen()**

Bentuk umum dari fungsi strlen() adalah sebagai berikut

```
Size_t strlen (const char* S);
```

Fungsi ini digunakan untuk mengembalikan panjang string dari S.

Contoh penggunaannya adalah sebagai berikut:

```
#include <iostream>
#include <cstring>

using namespace std;

int main() {
  char* S1;
  char* S2;
  int panjang_S1, panjang_S2;

 // Melakukan pengisian nilai terhadap variabel S1 dan S2
  S1 = "Budi";
  S2 = "Raharjo";

  panjang_S1 = strlen(S1);
  panjang_S2 = strlen(S2);

// Menampilkan nilai dari variabel panjang_S1 dan panjang_S2
  cout<<"Panjang S1 : "<<panjang_S1<<endl;
  cout<<"Panjang S2 : "<<panjang_S2<<endl;

  return 0;
}
```

Hasil program diatas adalah sebagai berikut:

```
Panjang S1 : 4
Panjang S2 : 7
```

g. Fungsi **strcmp()**

Bentuk umum dari fungsi strcmp() adalah sebagai berikut

```
Int strcmp (const char* S1, const char* S2);
```

Fungsi ini berguna untuk membandingkan string yang terdapat pada variabel S1 dan S2. Nilai yang akan dikembalikan yaitu:

1) 0 (nol) : hal ini terjadi jika S1 sama dengan S2
2) kurang dari 0 (negatif) : hal ini terjadi jika S1 lebih kecil dari S2
3) lebih dari 0 (positif) : hal ini terjadi jika S1 lebih besar dari S2

contoh penggunaannya adalah sebagai berikut:

```
#include <iostream>
#include <cstring>

using namespace std;

int main() {
  char* S1;
  char* S2;

  // Melakukan pengisian nilai yang sama
  // terhadap variabel S1 dan S2
  S1 = "COBA";
  S2 = "COBA";

  // Menampilkan hasil perbandingan string
  // dari variabel S1 dan S2
  cout<<"Hasil perbandingan : "<<strcmp(S1, S2)<<endl;

  // Melakukan perubahan nilai terhadap variabel S1
  S1 = "Coba";

  // Menampilkan kembali hasil perbandingan string
  // dari variabel S1 dan S2
  cout<<"Hasil perbandingan : "<<strcmp(S1, S2)<<endl;

  // Melakukan perubahan nilai terhadap variabel S2
  S2 = "coba";

  // Menampilkan kembali hasil perbandingan string
  // dari variabel S1 dan S2
  cout<<"Hasil perbandingan : "<<strcmp(S1, S2)<<endl;

  return 0;
}
```

Hasil program diatas adalah sebagai berikut:

```
Hasil perbandingan : 0
Hasil perbandingan : 32
Hasil perbandingan :-32
```

h. Fungsi **strrev()**

Bentuk umum dari fungsi strrev() adalah sebagai berikut

```
char*strrev (char* S);
```

Fungsi ini digunakan untuk mengembalikan string dari variabel S

```
#include <iostream>
#include <cstring>

using namespace std;

int main() {
  char* S = "Budi Raharjo";

  // Melakukan penulisan string mulai dari
  // posisi akhir variabel S
  strrev(S);

  // Menampilkan nilai dari variabel S setelah dibalik
  cout<< S<<endl;

  return 0;
}
```

Hasil program diatas adalah sebagai berikut:

```
ojrahaR iduB
```

i. Fungsi **strchr()**

Bentuk umum dari fungsi strchr() adalah sebagai berikut

```
char*strchr (const char* S, int ch);
```

Fungsi ini digunakan untuk mencari lokasi karakter dari suatu string. Jika karakter ditemukan, maka fungsi ini akan mengembalikan pointer dari string yang dimulai dari karakter tersebut, tetapi jika tidak maka fungsi akan mengembalikan nilai 0.

Contoh penggunaan program adalah

```
#include <iostream>
#include <cstring>

using namespace std;

int main() {
  char* S = "C++ adalah segalanya bagiku";
  char* PStr;

  // Melakukan pencarian karakter '+' di dalam varaibel S
  PStr = strchr(S, '+');

  // Menampilkan nilai dari variabel PStr
  cout<< PStr<<endl;

  return 0;
}
```

Hasil program diatas adalah sebagai berikut:

```
++ adalah segalanya bagiku
```

j. Fungsi **strstr()**

Bentuk umum dari fungsi strstr () adalah sebagai berikut

```
char*strstr (const char* S, const char* substr);
```

Fungsi ini akan mencari lokasi substring dari suatu string. Jika substring ditemukan, maka fungsi ini akan mengembalikan pointer dari string yang dimulai dari substring tersebut, tetapi jika tidak maka fungsi akan mengembalikan nilai 0.

```
#include <iostream>
#include <cstring>
using namespace std;
int main() {
  char* S = "Pencipta C++ adalah Bjarne Stroustrup";
  char* PStr;
// Melakukan pencarian substring "Bjarne" didalam varaibel S
  PStr = strstr(S, 'Bjarne');
  // Menampilkan nilai dari variabel PStr
  cout<< PStr<<endl;
  return 0;
}
```

2.6 Konversi String

Dalam membuat sebuah program, kerap kali kita dituntut untuk melakukan perubahan terhadap format string, misalnya mengubah string menjadi huruf kapital atau sebaliknya. Untuk melakukan hal ini, C++ menyediakan dua buah fungsi, yaitu strlwr () dan strupr ().

a. Fungsi strlwr ()

Bentuk umum dari fungsi strlwr () dapat dilihat dibawah ini

```
char*strlwr (char* S);
```

Fungsi ini akan mengubah string yang tersimpan dalam variabel S menjadi huruf kecil. Berikut ini adalah contoh penggunaannya dalam program

```
#include <iostream>
#include <cstring>

using namespace std;

int main() {
  char* S = "CONTOH MENGECILKAN HURUF";
  char* hasil;

  // Mengubah string dari variabel S menjadi huruf kecil
  hasil = strlwr(S);

  // Menampilkan nilai dari variabel hasil
  cout<< hasil<<endl;

  return 0;
}
```

Hasil program diatas adalah

```
contoh mengecilkan huruf
```

b. Fungsi strupr ()

Bentuk umum dari fungsi strupr () dapat dilihat dibawah ini

```
char*strupr (char* S);
```

Fungsi ini akan mengubah string yang tersimpan dalam variabel S menjadi huruf kapital. Berikut ini adalah contoh penggunaannya dalam program

```
#include <iostream>
#include <cstring>

using namespace std;

int main() {
  char* S = "contoh membesarkan huruf";
  char* hasil;

  // Mengubah string dari variabel S menjadi huruf keci
  hasil = strupr(S);

  // Menampilkan nilai dari variabel hasil
  cout<< hasil<<endl;

  return 0;
}
```

```
CONTOH MEMBESARKAN HURUF
```

BAB 3
OPERATOR

Sebagai langkah awal dalam memahami operator, ada baiknya kita mengetahui istilah-istilah yang berhubungan dengan operator. Sebagai contoh, jika terdapat suatu statemen yang berupa persamaan matematika dan dituliskan sebagai berikut:

```
C = 5+4
```

Maka:

C	disebut dengan *variabel*
=	disebut dengan *operator assignment*
5 dan 4	disebut dengan *operand*
5 + 4	disebut dengan *ekspresi*
+	disebut dengan *operator aritmetika (penambahan)*
C = 5 + 4	disebut dengan *statemen aritmatika*

3.1 Operator Assignment

Operator assignment adalah operator yang berfungsi untuk memasukkan *(assign)* nilai ke dalam suatu variabel ataupun konstanta. Operator ini dilambangkan dengan tanda **sama dengan** (=).

3.2 Operator Unary

Operator unary adalah operator yang hanya melibatkan sebuah operand. Adapun yang termasuk dalam kategori operator unary ini adalah sebagai berikut:

Operator	Jenis Operasi	Contoh
+	**Membuat nilai positif**	+7
-	**Membuat nilai negatif**	-7
++	**Increment**	C++
--	**Decrement**	C--

3.2.1 Increment

Increment adalah suatu penambahan nilai yang terjadi pada sebuah variabel. Adapun yang digunakan untuk melakukan *increment* adalah operator ++. Operator ini akan menambahkan nilai dari suatu variabel dengan nilai 1. Terdapat dua buah jenis *increment* yang terdapat dalam bahasa C++, yaitu *pre-increment* dan *post-increment.* Arti *pre-increment* yaitu melakukan penambahan nilai sebelum suatu variabel itu diproses, sedangkan *post-increment* yaitu melakukan proses terlebih dahulu sebelum dilakukan penambahan nilai. Adapun bentuk umum dari *pre-increment* dan *post-increment* dapat dilihat dibawah ini.

```
// Melakukan pre-increment
++nama_variabel;

// Melakukan post-increment
Nama_variabel++;
```

Contoh program

```
#include <iostream>

using namespace std;
int main() {
  int C;// Mendeklarsikan variabel C
  // Mengisikan nilai ke dalam variabel C dengan nilai 5
  C = 5;
  // Melakukan pre-increment
  cout<<"Nilai C awal        : "<<C<<endl;
  cout<<"Nilai ++C           : "<<++C<<endl;
  cout<<"Nilai C akhir       : "<<C<<endl;
  cout<<'\n';

  // Mengubah nilai yang terdapat dalam variabel C dengan nilai 10
  C = 10;
  // Melakukan post-increment
  cout<<"Nilai C awal              : "<<C<<endl;
  cout<<"Nilai C++                 : "<<C++<<endl;
  cout<<"Nilai C akhir             : "<<C<<endl;
  return 0;
}
```

Hasil dari program diatas adalah

Pre increment	Nilai C awal : 5	*Post-increment*	Nilai C awal : 10
	Nilai ++C : 6		Nilai C++ : 10
	Nilai C akhir : 6		Nilai C akhir : 11

3.2.2 Decrement

Decrement adalah suatu pengurangan nilai yang terjadi pada sebuah variabel. Adapun yang digunakan untuk melakukan *increment* adalah operator --. Operator ini akan mengurangkan nilai dari suatu variabel dengan nilai 1. Terdapat dua buah jenis *decrement* yang terdapat dalam bahasa C++, yaitu *pre-increment* dan *post-increment.* Arti *pre-increment* yaitu melakukan pengurangan nilai sebelum suatu variabel itu diproses, sedangkan *post-increment* yaitu melakukan proses terlebih dahulu sebelum dilakukan pengurangan nilai. Adapun bentuk umum dari *pre-increment* dan *post-increment* dapat dilihat dibawah ini.

```
#include <iostream>
using namespace std;

int main() {
  int C; // Mendeklarsikan variabel C

  // Mengisikan nilai ke dalam variabel C dengan nilai 5
  C = 5;

  // Melakukan pre-increment
  cout<<"Nilai C awal          : "<<C<<endl;
  cout<<"Nilai --C             : "<<--C<<endl;
  cout<<"Nilai C akhir         : "<<C<<endl;
  cout<<'\n';

  // Mengubah nilai yang terdapat dalam variabel C
  // dengan nilai 10
  C = 10;

  // Melakukan post-decrement
  cout<<"Nilai C awal          : "<<C<<endl;
  cout<<"Nilai C--             : "<<C--<<endl;
  cout<<"Nilai C akhir         : "<<C<<endl;

  return 0;
}
```

Hasil dari program diatas adalah

Pre increment	Nilai C awal : 5	*Post-increment*	Nilai C awal : 10
	Nilai ++C : 4		Nilai C++ : 10
	Nilai C akhir : 4		Nilai C akhir : 9

3.3 Operator Binary

Opertaor binary adalah operator yang digunakan dalam operasi yang melibatkan dua buah *operand.* Dalam bahasa C++, operator *binary* ini dikelompokkan menjadi empat jenis, yaitu operator aritmetika, logika, relasional dan bitwise.

3.3.1 Operator Aritmetika

Operator aritmatika adalah operator yang digunakan untuk melakukan operasi-operasi aritmetika. Adapun yang termasuk dalam operator aritmetika di dalam C++ adalah seperti berikut ini.

Operator	Jenis Operasi	Contoh
+	Penjumlahan	1 + 1 = 2
-	Pengurangan	2 - 1 = 1
*	Perkalian	1 * 1 = 1
/	Pembagian	9 / 3 = 3
%	Sisa bagi (modulus)	10 3 = 1

3.3.2 Operator Logika

Operator logika adalah operator yang digunakan untuk melakukan operasi dimana nilai yang dihasilkan dari operasi tersebut hanya berupa nilai benar (*true*) dan salah (*false*). Adapun yang termasuk dalam operator logika adalah seperti berikut.

Operator	Jenis Operasi	Contoh
&&	AND (dan)	1 && 1 = 1
||	OR (atau)	1 || 0 = 1
!	NOT (negasi)	!0 = 1

3.4 Operator Relasional

Operator relasional adalah operator yang digunakan untuk menentukan relasi atau hubungan dari dua buah *operand*. Operator ini ditempatkan di dalam sebuah ekspresi, yang kemudian akan menentukan benar atau tidaknya sebuah ekspresi. Yang termasuk dalam kategori operasi relasional adalah sebagai berikut:

Operator	Jenis Operasi	Contoh
>	Lebih besar	(3 > 2) = 1
<	Lebih kecil	(3 < 2) = 0
>=	Lebih besar atau sama dengan	(3 >= 3) = 1
<=	Lebih kecil atau sama dengan	(3 <= 2) = 0
==	Sama dengan	(5 == 2) = 0
!=	Tidak sama dengan	(5 != 2) = 1

3.5 Operator Bitwise

Operator yang digunakan untuk melakukan operasi-operasi yang berhubungan dengan pemanipulasian bit. Operator bitwise ini hanya dapat dilakukan pada operand yang bertipe char dan int saja karena ini berkoresponden dengan tipe byte atau word di dalam bit. Adapun yang termasuk dalam operator bitwise dalam C++ adalah sebagai berikut:

Operator	Jenis Operasi	Contoh
&	AND	1 & 0 = 0
|	OR	1 | 0 = 1
^	EXCLUSIVE OR (XOR)	1 ^ 1 = 0
~	NOT	~1 = 0
>>	Shift Right	16 >> 1 = 8
<<	Shift Left	1 << 2 = 4

3.6 Operator Ternary

Operator ternary adalah operator yang digunakan dalam operasi yang melibatkan tiga buah *operand.* Adapun operator yang digunakan untuk menyatakannya adalah operator ***?:***. Bentuk umum dari penggunaan operator ternary adalah

```
Ekspresi1 ? Ekspresi2 :Ekspresi3;
```

Jika ekspresi1 bernilai benar,maka program akan mengeksekusi ekspresi2. Sedangkan jika ekspresi1 bernilai salah, maka yang dieksekusi adalah ekpresi3.

BAB 4
PERCABANGAN

Di dalam C++, terdapat dua buah jenis struktur yang digunakan untuk mengimplementasikan suatu percabangan, yaitu dengan menggunakan **if** dan struktur **switch.** Untuk lebih memahami konsep percabangan, perhatikan kalimat dibawah ini.

> "Jika Andi lulus ujian maka Andi akan dibelikan sepeda motor oleh ayahnya."

Coba anda amati, pada kalimat diatas yang merupakan kondisi adalah *lulus ujian.* Pada kasus ini sepeda motor hanya akan dibeli jika Andi lulus ujian. Sebaliknya, jika tidak lulus, maka sepeda motor pun tidak akan dibeli. Suatu percabangan dapat terdiri dari satu kondisi maupun lebih.

4.1 Struktur Satu Kondisi

Struktur ini merupakan struktur yang paling sederhana karena hanya melibatkan satu buah ekspresi yang akan dipelajari. Bentuk umum dari struktur percabangan yang memiliki satu kondisi adalah sebagai berikut:

```
// Jika terdapat lebih dari satu statemen

if (kondisi) {
        Statemen1;
        Statemen2;
        ...
}
// jika hanya terdapat satu statemen, dapat ditulis
// seperti dibawah ini

if (kondisi) Statemen;
```

Sebagai contoh, perhatikan program berikut:

```
#include <iostream>

using namespace std;

int main() {
  int nilai;

  // Memberikan informasi agar user memasukkan
  // sebuah bilangan bulat
  cout<<"Masukkan sebuah bilangan bulat : ";

  // Membaca nilai yang dimasukkan dari keyboard dan disimpan
  // ke variabel nilai
  cin>>nilai;

  // Menampilkan sebuah teks jika nilai yang tersimpan
  // lebih besar dari nol
  if (nilai > 0)
    cout<<"Nilai yang Anda masukkan adalah bilangan positif";

  return 0;
}
```

Hasil dari program diatas bersifat dinamis, artinya tidak setiap proses eksekusi program akan memberikan hasil yang sama, karena hasilnya tentu tergantung kepada nilai yang dimasukkan oleh user.

4.2 Struktur Dua Kondisi

Struktur ini merupakan struktur yang kompleks. Dalam struktur ini terdapat statemen khusus yang berguna untuk mengatasi kejadian apabila kondisi yang didefinisikan tersebut tidak terpenuhi (bernilai salah). Bentuk umum dari struktur percabangan ini adalah sebagai berikut:

```
if (kondisi) {
        Statemen_jika kondisi_terpenuhi;
} else {
        Statemen_jika kondisi_tidak_terpenuhi;
}
```

Contoh program dari struktur dua kondisi adalah sebagai berikut:

```
#include <iostream>

using namespace std;

int main() {
  int bilangan;

  cout<<"Masukkan bilangan bulat yang akan diperiksa : ";
  cin>>bilangan;

  // Melakukan pengecekan bilangan apakah habis
  // dibagi dua atau tidak
  if (bil % 2 == 0) {
    cout<<bil<<" adalah bilangan genap";
  } else {
    cout<<bil<<" adalah bilangan ganjil";
  }

  return 0;
}
```

Hasil dari contoh program diatas adalah

```
Masukkan bilangat bulat yang akan diperiksa :7
7 adalah bilangan ganjil
```

Coba amati hasil diatas, pada saat program dijalankan user memasukkan bilangan 7 dan nilai tersebut akan disimpan ke dalam variabel **bil.** Selanjutnya program akan melakukan pengecekan terhadap nilai yang terdapat dalam variabel tersebut melalui ekspresi (bil % 2 ==0). Maksud dari ekspresi tersebut adalah melakukan pengecekan apakah nilai dari variabel tersebut habis dibagi dua (*sisa =0*) atau tidak. Jika ya, maka bilangan tersebut tentu akan termasuk kedalam bilangan genap. Namun pada kasus ini karena 7 tidak dapat habis dibagi dua maka program akan memilih statemen yang terdapat pada bagian **else.**

4.3 Struktur Tiga Kondisi

Struktur ini merupakan perluasan struktur yang memiliki dua kondisi diatas, yaitu dengan menyisipkan (menambhakan) satu atau lebih kondisi kedalamnya. Bentuk umum dari struktur percabangan yang memiliki lebih dari dua kondisi adalah sebagai berikut:

```
if (kondisi1) {
        Statemen_jika kondisi1_terpenuhi;
} else if (kondisi2) {
        Statemen_jika kondisi2_terpenuhi;
} else if (kondisi3) {
        Statemen_jika kondisi3_terpenuhi;
}
...
else {
        Statemen_jika_semua_kondisi_diatas_tidak_terpenuhi;
}
```

Sebagai contoh untuk menerapkan konsep ini adalah pada saat menentukan sebuah bilangan bulat, apakah bilangan tersebut termasuk bilangan positif, negatif atau nol. Adapaun sintaks programnya dapat dilihat dibawah ini.

```
#include <iostream>

using namespace std;

int main() {
  int bil;

  cout<<"Masukkan sebuah bilangan bulat yang akan diperiksa : ";
  cin>>bil;

  if (bil > 0) {
    cout<<bil<<" adalah bilangan POSITIF";
  } else if (bil < 0) {
    cout<<bil<<" adalah bilangan NEGATIF";
  } else {
    cout<<"Anda memasukkan bilangan NOL";
  }

  return 0;
}
```

Hasil yang diberikan pada program diatas adalah sebagai berikut:

```
Masukkan sebuah bilangan yang akan diperiksa : -8
-8 adalah bilangan NEGATIF
```

Contoh lain penggunaan statemen **if** untuk membuat program yang dapat menentukan nilai indeks (A,B,C,D dan E) dari nilai akhir yang didapatkan oleh seorang mahasiswa. Untuk menjawab kasus ini, kita mempunyai kriteria penilaian, yaitu sebagai berikut:

```
A    : nilai ≥ 85
B    : 70 ≤ nilai ≥ 85
C    : 55 ≤ nilai ≥ 70
D    : 40 ≤ nilai ≥ 55
E    : nilai < 40
```

Dengan mengacu kriteria diatas, maka kita dapat menuliskan sintaks dari program tersebut di bawah ini.

```
#include <iostream>
using namespace std;
int main() {
  double nilai;
  char indeks;
  // Meminta user untuk menentukan nilai
  // yang diperoleh dalam bentuk bilangan
cout<<"Masukkan nilai yang diperoleh : ";
cin>>nilai;
  // Melakukan konversi nilai menjadi nilai indeks
  if (nilai >= 85) {
    indeks = 'A';
  } else if (nilai >= 70) {
    indeks = 'B';
  } else if (nilai >= 55) {
    indeks = 'C';
  } else if (nilai >= 40) {
    indeks = 'D';
  } else {
    indeks = 'E';
  }
  // Menampilkan nilai indeks yang didapatkan
  cout<<"Nilai indeks dari nilai "<<nilai<<" adalah "<<indeks;
return 0;
}
```

Contoh hasil dari program diatas adalah sebagai berikut:

```
Masukkan nilai yang diperoleh : 77
Nilai indeks dari nilai 77 adalah B
```

4.4 Pemilihan dengan Switch

Selain menggunakan struktur **if,** C++ juga menawarkan bentuk pemilihan dengan menggunakan kata **switch.** Bentuk umum penulisan strutur tersebut adalah sebagai berikut.

```
Switch (ekspresi) {
        case nilai_konstan1 : Statemen_statemen; break;
        case nilai_konstan2 : Statemen_statemen; break;
        ...
        case nilai_konstanN : Statemen_statemen; break;
        default
        Statemen_statemen_alternatif;
}
```

Tipe data dari ekspresi diatas haruslah bilangan bulat atau karakter. Selain tipe tersebut, C++ tidak mengizinkan. Statemen **default** berguna untuk mengeksekusi statemen alternatif, yaitu jika nilai yang masukkan tidak sesuai dengan nilai-nilai konstan yang telah didefinisikan. Kita dapat mendefinisikan nilai konstan tersebut menggunakan statemen **case.** Berikut contoh program menggunakan struktur **switch** untuk menentukan nama hari dari nilai bilangan yang dimasukkan.

```
#include <iostream>
using namespace std;
int main() {
 int bil;
 cout<<"Masukkan sebuah bilangan (1..7) : "; cin>>bil;
 switch (bil) {
 case 1 : cout<<"Hari ke-"<<bil<<" : adalah MINGGU";break;
 case 2 : cout<<"Hari ke-"<<bil<<" : adalah SENIN"; break;
 case 3 : cout<<"Hari ke-"<<bil<<" : adalah SELASA";break;
 case 4 : cout<<"Hari ke-"<<bil<<" : adalah RABU"; break;
 case 5 : cout<<"Hari ke-"<<bil<<" : adalah KAMIS";break;
 case 6 : cout<<"Hari ke-"<<bil<<" : adalah JUMAT";break;
 case 7 : cout<<"Hari ke-"<<bil<<" : adalah SABTU";break;
 default : cout<<"Tidak terdapat nama hari ke-"<<bil;
 }
```

Contoh hasil yang diberikan dalam program diatas adalah sebagai berikut:

```
Masukkan sebuah bilangan (1...7) : 5
Hari ke-5 adalah KAMIS
```

Apa yang terjadi jika kita memasukkan nilai yang lebih kecil dari 1 atau lebih besar dari 7? Tanpa menjalankan program tersebut, kita sudah mengetahui bahwa yang akan dieksekusi adalah statemen yang terdapat pada pilihan **default.** Untuk membuktikan hal ini, kita masukkan nilai 9, maka program diatas akan memberikan hasil seperti dibawah ini.

```
Masukkan sebuah bilangan (1...7) : 9
Tidak terdapat nama hari ke-9
```

BAB 5
PENGULANGAN

Pengulangan adalah suatu proses yang melakukan statemen-statemen dalam sebuah program secara terus menerus sampai terdapat kondisi untuk menghentikannya. Struktur pengulangan akan sangat membantu dalam efisiensi program. Dalam bahasa C++, terdapat tiga buah jenis struktur pengulangan yaitu struktur **for**, struktur **while,** struktur **do-while**.

5.1 Struktur *for*

Struktur jenis ini digunakan untuk melakukan pengulangan yang telah diketahui banyaknya. Jenis ini merupakan jenis struktur pengulangan yang paling mudah dipahami. Untuk melakukan pengulangan dengan menggunakan struktur ini, kita harus memiliki sebuah variabel sebagai indeksnya. Bentuk umum struktur **for** adalah sebagai berikut:

```
// Untuk pengulangan yang sifatnya menaik (increment)
for (variabel = nilai_awal; kondisi; variabel++){
        statemen_yang_akan_diulang;
}

// Untuk pengulangan yang sifatnya menurun (decrement)
for (variabel = nilai_awal; kondisi; variabel--){
        statemen_yang_akan_diulang;
}
```

Contoh program pengulangan menaik dan menurun

```
#include <iostream>

using namespace std;

int main() {
  cout<<"PENGULANGAN MENAIK"<<endl;
  for (C=0; C<10; C++) {
    cout<<C+1<<endl;
  }
```

Lanjutan program

```
// Membuat spasi vertikal
  cout<<'\n';  // dapat ditulis cout<<endl;
  cout<<"PENGULANGAN MENURUN"<<endl;
  for (J=10; J>0; J--) {
    cout<<J<<endl;
  }

  return 0;
}
```

Hasil yang akan diberikan dari program diatas adalah sebagai berikut:

```
PENGULANGAN MENAIK
1
2
3
4
5
6
7
8
9
10

PENGULANGAN MENURUN
1
2
3
4
5
6
7
8
9
10
```

5.2 Struktur *while*

Struktur pengulangan jenis ini adalah pengulangan yang melakukan pengecekan kondisi awal blok struktur. Bentuk umum dari struktur **while** adalah sebagai berikut:

```
While (kondisi) {
        Statemen_statemen_yang_akan_diulang;
}
```

Contoh program menggunakan struktur **while**

```
#include <iostream>

using namespace std;

int main() {
 int C;          // Mendeklarasikan varaiebl C sebagai
            // indeks pengulangan

 C = 0;                  // Melakukan inisialisasi nilai terhadap
            // variabel C

 while (C<10) {
   cout<<"Saya sangat menyukai C++"<<endl;
   C++;        /* Statemen ini berguna untuk menaikkan nilai,
          dan setelah bernilai 10,
          maka pengulangan akan dihentikan */
 }

 return 0;
}
```

Hasil dari program diatas adalah sebagai berikut:

```
Saya sangat menyukai C++
Saya sangat menyukai C++
Saya sangat menyukai C++
Saya sangat menyukai C++
Saya sangat menyukai C++
Saya sangat menyukai C++
Saya sangat menyukai C++
Saya sangat menyukai C++
Saya sangat menyukai C++
Saya sangat menyukai C++
```

5.3 Kata Kunci break dan continue

Kata kunci ini berfungsi untuk menghentikan perulangan dan melanjutkan ke perintah-perintah yang lain.

5.4 Struktur *do-while*

Pada struktur *do-while* kondisi pengecekan ditempatkan di bagian akhir. Hal ini menyebabkan struktur pengulangan ini minimal akan melakukan satu kali proses walaupun kondisi yang didefinisikan tidak terpenuhi (bernilai salah). Bentuk umum dari struktur *do-while*

```
do {
        statemen_yang_akan_diulang;
}       while (kondisi);
```

Contoh program menggunakan struktur **do-while.**

Misalnya kita memasukkan dua buah bilangan bulat yaitu 8 dan 12, maka faktor persekutuan terbesar dari kedua bilangan tersebut adalah 4. Perhatikan tabel di bawah ini

Bilangan	Faktor
8	1, 2, **4**, 8
12	1, 2, 3, **4**, 6, 12

Contoh program penggunaan strukutr *do-while* . Seperti yang kita lihat diatas bahwa faktor yang sama dan paling besar dari bilangan 8 dan 12 adalah **4.** Jika dibuat menggunakan struktur do-while, maka sintaks programnya adalah sebagai berikut:

```
#include <iostream>

using namespace std;
int main() {
  int Bil1, Bil2;
  int sisa;
  cout<<"Masukkan bilangan pertama  : "; cin>>Bil1;
  cout<<"Masukkan bilangan kedua    : "; cin>>Bil2;
  // Melakukan pertukaran nilai
  if (Bil1 < Bil2) {
    int temp = Bil1;
    Bil1 = Bil2;
    Bil2 = temp;   }
  do {
    sisa = Bil1 % Bil2;
    Bil1 = Bil2;
    Bil2 = sisa;}
        while (sisa != 0);
  cout<<"\nFaktor persekutuan terbesar = "<<Bil1;
  return 0;
}
```

Contoh hasil program diatas adalah diperoleh sebagai berikut:

```
Masukkan bilangan pertama : 8
Masukkan bilangan kedua : 12

Faktor persekutuan terbesar = 4
```

BAB 6

POINTER DAN REFERENCE

6. 1. Pointer

Pointer dapat didefinisikan sebagai suatu variabel yang menyimpan alamat memori. Cara memasukkan nilai pada pointer harus memperhatikan tipe data dari pointer itu sendiri dengan tipe data dari variabel yang akan ditempatinya. Sebagai contoh kita akan mendeklerasikan pointer **P** ke tipe **double** dan kita juga memiliki variabel **X** yang bertipe **int.** Pada kasus ini kita tidak diizinkan untuk menyimpan alamat memori dari variabel **X** ke pointer **P** karena tipenya berbeda.

Untuk membuat pointer yang kita deklerasikan dapat menunjuk ke semua tipe data, yaitu dengan mendeklerasikan poniter tersebut sebagai pointer tanpa tipe atau disebut **void pointer.** Bentuk umum tipe void pointer adalah sebagai berikut:

```
Void *nama_pointer;
```

Setiap pointer yang kita deklerasikan, maka pointer tersebut akan menunjuk lokasi acak di memori. Oleh karena itu kita harus mengeset ointer yang kita deklerasikan tersebut dalam keadaan NULL,atau tidak menunjuk lokasi apapun. Contoh program memakai **NULL** adalah sebagai berikut

```
#include <iostream>

using namespace std;

int main() {
  int *P;           // Mendeklarasikan pointer P
          // yang akan menunjuk tipe data int

  cout<<"Alamat yang ditunjuk oleh pointer P : "<<P;

  return 0;
}
```

Maka hasil yang akan diperoleh adalah sebagai berikut:

```
Alamat yang ditunjuk oleh pointer P : 00000000
```

6. 2. Reference

Reference digunakan untuk membuat alias atau nama lain (julukan) dari sebuah variabel. Untuk membuat **reference** adalah menggunakan tanda **&** dibelakang tipe data yang akan diacu. Bentuk umum dari pembuatan reference adalah

```
Tipe_data&nama_alias =nama_variabel;
```

Contoh program **reference**

```
#include <iostream>

using namespace std;

int main() {

 int X; // Mendeklarasikan variabel X

 // Membuat alias dari variabel X dengan nama AliasX
 int& AliasX = X;

 // Mengisikan nilai ke dalam variabel X
 X = 5;

 // Menampilkan nilai yang disimpan dalam variabel X dan AliasX
 cout<<"Nilai X      : "<<X<<endl;
 cout<<"Nilai AliasX         : "<<AliasX<<endl;
 cout<<endl;

 // Mengisikan nilai ke dalam AliasX
 AliasX = 26;

 // Menampilkan kembali nilai yang disimpan dalam variabel X
 // dan AliasX
 cout<<"Nilai X      : "<<X<<endl;
 cout<<"Nilai AliasX         : "<<AliasX<<endl;

 return 0;
}
```

Hasil yang akan diperoleh dari program diatas adalah

```
Nilai X          : 5          Nilai AliasX  : 5
Nilai X          : 26         Nilai AliasX  : 26
```

BAB 7
ARRAY

Array adalah sebuah variabel yang menyimpan sekumpulan data yang memiliki tipe sama. Setiap data tersebut menempati lokasi atau alamat memory yang berbeda-beda dan selanjutnya disebut dengan elemen array. Elemen array itu diakses melalui indeks yang terdapat didalamnya. Dalam C++, indeks array selalu dimulai dari **0,** bukan 1. Berikut ilustrasi sebuah array

Nilai ke-1	**Nilai ke-2**	**.....**	**Nilai ke-N**	Nilai elemen array
Alamat ke-1	Alamat ke-2	.....	Alamat ke-N	Alamat elemen array
0	1	.....	N	Indeks elemen array

Untuk mendeklerasikan sebuah array adalah menggunakan tanda **[]** (*bracket*). Bentuk umum dari pendeklerasiannya adalah sebagai berikut:

```
tipe_data nama_array[jumlah_elemen];
```

Sebagai contoh jika kita ingin mendeklarasikan sebuah array (misalnya dengan nama LARIK) yang memiliki 10 elemen dengan tipe data int, maka pendeklarasiaannya adalah seperti yang tampak dibawah ini:

```
int LARIK [10];
```

Ruang memori yang dibutuhkan untuk pendeklarasian array tersebut adalah 100 byte, yang berasal dari 25 x 4 byte (4 merupakan ukuran dari tipe data int). Sedangkan cara yang digunakan untuk mengakses elemennya adalah dengan menuliskan indeksnya. Misalnya kita ingin mengambil nilai yang terdapat pada elemen ke-10 dan menampung nilai tersebut ke dalam sebuah variabel yang bertipe int juga (misal X), maka kita harus menuliskan sintaks di bawah ini.

```
X=LARIK [9];
```

Kenapa 9, bukan 10? Ingat, indeks array selalu dimulai deri 0 sehingga untuk mengakses elemen ke-10, maka indeks yang kita butuhkan adalah 10-1, yaitu 9.

7.1 Mengisikan Nilai ke dalam Elemen Array

Untuk mengisikan nilai ke dalam elemen-elemen array, kita dapat melakukannya langsung untuk setiap elemen, misalnya seperti berikut.

```
A[0] = 1
A[1] = 2
A[2] = 3
…
dst
```

Cara ini tidak direkomendasikan karena tidak efisien. Cara yang lebih umum dan banyak digunakan oleh para programmer untuk mengisikan ilai ke dalam elemen-elemen array adalah dengan menggunakan pengulangan (*looping*). Cara ini akan jauh lebih cepat dibandingkan cara manual seperti di atas. Sebagai contoh jika kita ingin melakukan pengisian 25 elemen array, maka kita dapat menuliskan sintaks seperti di bawah ini.

```
for (int C=0; C<25; C++) {
count <<”A[“<<C<<”] = “; cin>>A[C];}
```

Contoh programnya adalah sebagai berikut:

```
#include <iostream>
using namespace std;

int main() {

 // Mendeklarasikan array A dengan 5 buah elemen bertipe int
 int A[5];

 // Mengisikan nilai ke dalam elemen array
 cout<<"Masukkan nilai yang diinginkan"<<endl;
 for (int C=0; C<5; C++) {
   cout<<"A["<<C<<"] = "; cin>>A[C];
 }
```

7.2 Menampilkan Nilai yang Terdapat pada Array

Setelah memahami cara mengisikan nilai ke dalam elemen array, sekarang kita akan membahasa bagaimana cara untuk mengakses atau menampilkan nilai-nilai tersebut. Konsepnya sama seperti di atas, kita akan menggunakan pengulangan untuk menampilkan nilai yang terdapat pada elemen array.

```
#include <iostream>
using namespace std;

int main() {
  // Mendeklarasikan array A dengan 5 buah elemen bertipe int
  int A[5];
  // Mengisikan nilai ke dalam elemen array
  cout<<"Masukkan nilai yang diinginkan"<<endl;
  for (int C=0; C<5; C++) {
    cout<<"A["<<C<<"] = "; cin>>A[C];
  }
  cout<<'\n';
  // Menampilkan nilai yang terdapat dalam elemen array
  cout<<"Menampilkan nilai yang telah dimasukkan"<<endl;
  for (int J=0; J<5; J++) {
    cout<<"Nilai yang terdapat pada elemen ke-";
    cout<<J+1<<" : "<<A[J]<<endl;
  }
  return 0;
}
```

Contoh hasil yang akan diberikan dari program di atas adalah sebagai berikut:

```
Masukkan nilai yang diinginkan
A [0] = 5
A [1] = 10
A [2] = 15
A [3] = 20
A [4] = 25
Menampilkan nilai yang telah dimasukkan
Nilai yang terdapat pada elemen ke -1 : 5
Nilai yang terdapat pada elemen ke -2 : 10
Nilai yang terdapat pada elemen ke -3 : 15
Nilai yang terdapat pada elemen ke -4 : 20
Nilai yang terdapat pada elemen ke -5 : 25
```

7. 3. Melakukan Inisialisasi Array

Pada saat kita mendeklarasikan sebuah array, kita dapat langsung melakukan inisialisasi nilai terhadap elemen-elemen untuk mengisikan nilai *default* pada elemen array sehingga jika elemen bersangkutan tidak diisi dengan nilai baru, maka nilai yang digunakan adalah nilai yang telah ada. Adapun bentuk umum dari inisialisasi array adalah seperti yang tampak di bawah ini.

```
tipe, data nama_array [N] = {nilai1, nilai2, …, nilaiN } ;
```

Berikut ini adalah contoh program yang menunjukkan proses inisialisasi nilai pada elemen-elemen array.

```
#include <iostream>

using namespace std;

int main() {
  // Mendeklarasikan array
  // dan langsung menginisialisasi nilainya
  int A[5] = { 10, 20, 30, 40, 50 };

  // Menampilkan nilai yang terdapat pada elemen array
  cout<<"Sebelum dilakukan perubahan nilai"<<endl;
  cout<<"A[0] = "<<A[0]<<endl;
  cout<<"A[1] = "<<A[1]<<endl;
  cout<<"A[2] = "<<A[2]<<endl;
  cout<<"A[3] = "<<A[3]<<endl;
  cout<<"A[4] = "<<A[4]<<endl;

  // Mengubah elemen ke-1 dan ke-2
  A[0] = 12;
  A[1] = 25;

  // Menampilkan kembali nilai yang terdapat pada elemen array
  cout<<"Setelah dilakukan perubahan nilai"<<endl;
  cout<<"A[0] = "<<A[0]<<endl;
  cout<<"A[1] = "<<A[1]<<endl;
  cout<<"A[2] = "<<A[2]<<endl;
  cout<<"A[3] = "<<A[3]<<endl;
  cout<<"A[4] = "<<A[4]<<endl;

  return 0;
}
```

Hasil yang akan diberikan dari program di atasa adalah sebagai berikut:

```
Sebelum dilakukan perubahan nilai
10
20
30
40
50
Setelah dilakukan perubahan nilai
12
25
30
40
50
```

Seperti yang kita lihat di atas bahwa elemen ke-3, 4 dan 5 tidak dilakukan perubahan sehingga nilai yang digunakan adalah nilai default.

7.4 Melakukan Pencarian pada Elemen Array

Salah satu permasalahan yang sering muncul pada saat kita menggunakan array adalah tuntutan untuk melakukan pencarian elemen array. Misalnya di jurusan elektro dari perguruan tinggi tertentu terdapat sekumpulan data mahasiswa yang disimpan dalam sebuah array, kemudian kita akan melakukan pencarian data mahasiswa yang bernama Anang Hermansyah dari array tersebut. Contoh lainnya adalah penacrian data rekening bank, serta masih banyak lagiyang lainnya. Kasus-kasus semacam ini banyak dijumpai jika kita telah berhubungan dengan data. Maka dari itu, pada bagian ini kita akan membahas bagaimana cara melakukan pencarian elemen tertentu pada sebuah array.

Berikut ini contoh program yang akan melakukan pencarian nilai dari sekelompok data yang bertipe int.

```
#include <iostream>

using namespace std;

int main() {
  // Mendeklarasikan array
  // dengan melakukan inisialisasi nilai ke dalamnya
  int A[10] = { 12, 24, 13, 25, 10, 13, 21, 20, 15, 18 };
  int BIL;        // Variabel untuk menampung nilai yang akan
dicari

  // Menampilkan nilai yang terdapat
  // pada elemen-elemen array di atas
  for (int C=0, C<10; C++) {
    cout<<A[C]<<endl;
  }
  cout<<endl;

  // Memasukkan nilai yang akan dicari
  cout<<"Masukkan nilai yang akan dicari : "; cin>>BIL;

  // Melakukan pencarian data
  for (int J=0; J<10; J++) {
    if (A[J] == BIL) {
      cout<<"Nilai yang dicari terdapat pada indeks ke-"<<J;
      break;
    }
  }
```

Contoh hasil yang diberikan dari program di atas adalah sebagai berikut:

```
12
24
14
25
10
13
21
20
15
18
Masukkan nilai yang akan dicari : 13
Nilai yang dicari terdapat pada indeks ke-5
```

Sebenarnya nilai 13 ditemukan pada elemen array ke-6, namun larena indeksnya dimulai dari 0, maka indeks yang akan diberikan untuk data tersebut adalah 5.

7.5 Mengurutkan Elemen Array

Setelah kita mempelajari algoritma, maka sebenarnya kita telah mengetahui bahwa aelemen larik dapat diurutkan dengan beberapa buah metode, diantaranya metode gelembung (*bubble sort*), maksimum-minimum (*maximum-minimum sort*), sisip (*insertion sort*), *heap sort,* dan masih banyak lagi yang lainnya. Salah satu dari kegunaan suatu pengurutan data adalah usaha mempermudah dan mempercepat proses pencapaian data. Untuk lebih memahaminya, perhatikan dua buah program di bawah ini yang masing-masing akan mengurutkan elemen array dengan metode gelembung dan maksimum-minimum.

```
#include <iostream>

using namespace std;

int main() {

 // Mendeklarasikan array dengan 7 buah elemen
 // yang bertipe int
 int A[7];

 // Mendeklarasikan variabel-variabel bantu yang diperlukan
 int j, k, C, temp;

 // Memasukkan nilai array
 cout<<"Masukkan nilai pada elemen array :"<<endl;
 for (C=0; C<7; C++) {
   cout<<"A["<<C<<" ] = "; cin>>A[C];
 }

 // Menampilkan nilai sebelum diurutkan
 cout<<"\nNilai elemen array sebelum diurutkan :"<<endl;
 for (C=0; C<7; C++) {
   cout<<"A["<<C<<"] = "<<A[C]<<endl;
 }
```

Mengurutkan elemen array dengan metode gelembung

Contoh hasil yang akan diberikan dari program di atas adalah sebagai berikut:

```
Masukkan nilai pada elemen array:
A [0] = 70
A [1] = 10
A [2] = 40
A [3] = 20
A [4] = 30
A [5] = 60
A [6] = 50
Nilai elemen array sebelum diurutkan:
A [0] = 70
A [1] = 10
A [2] = 40
A [3] = 20
A [4] = 30
A [5] = 60
A [6] = 50
Nilai elemen array setelah diurutkan:
A [0] = 10
A [1] = 20
A [2] = 30
A [3] = 40
A [4] = 50
A [5] = 60
A [6] = 70
```

7.6 Menggunakan Metode Maksimum-minimum

```
#include <iostream>
using namespace std;

int main() {

 // Mendeklarasikan array dengan 7 buah elemen
 // yang bertipe int
 int A[7];

 // Mendeklarasikan variabel-variabel bantu yang diperlukan
 int j, k, C, temp;

 // Memasukkan nilai array
 cout<<"Masukkan nilai pada elemen array :"<<endl;
 for (C=0; C<7; C++) {
   cout<<"A["<<C<<" ] = "; cin>>A[C];
 }
```

Contoh hasil yang akan diberikan dari program di atas adalah sebagai berikut:

```
Masukkan nilai pada elemen array:
A [0] = 6
A [1] = 3
A [2] = 7
A [3] = 2
A [4] = 5
A [5] = 1
A [6] = 4
Nilai elemen array sebelum diurutkan:
A [0] = 6
A [1] = 3
A [2] = 7
A [3] = 2
A [4] = 5
A [5] = 1
A [6] = 4
Nilai elemen array setelah diurutkan:
A [0] = 1
A [1] = 2
A [2] = 3
A [3] = 4
A [4] = 5
A [5] = 6
A [6] = 7
```

7.7. Array yang bersifat Konstan

Nilai dalam elemen array dapat dibuat tetap, yaitu dengan mendefinisikannya sebagai konstanta. Caranya yaitu dengan menggunakan kunci **const** di depan nama nama array yang didefinisikannya. Berikut ini contih program yang menunjukkan hal tersebut.

```
#include <iostream>
using namespace std;
int main() {
 // Mendeklarasikan array yang bersifat konstan
 const int A[5] = { 10, 20, 30, 40, 50 };
 // Menampilkan nilai yang terdapat pada array A
 for (int C=0; C<5; C++) {
  cout<<"A["<<C<<"] = "<<A[C]<<endl;
 }  return 0;}
```

Hasil yang akan diberikan dari program di atas adalah sebagai berikut:

```
Masukkan nilai pada elemen array:
A [0] = 10
A [1] = 20
A [2] = 30
A [3] = 40
A [4] = 50
```

7.8 Array Sebagai Tipe Data Bentukan

Dalam C++, array juga dapat digunakan sebagai tipe data bentukan seperti halnya struktur dan enumerasi. Untuk melakukan hal ini kita harus menggunakan kata kunci *typedef,* dimana berfungsi untuk memberikan nama lain dari array yang dideklarasikan. Berikut ini bentuk umum untuk membuat array sebagai tipe data bentukan.

```
typedef type_data_nama_array [jumlah elemen];
```

Untuk lebih memahaminya, perhatikan contoh program di bawah ini dimana terdapat sebuah array yang didefinisikan sebagai tipe data bentukan.

```
#include <iostream>
using namespace std;

int main() {

  // Mendeklarasikan tipe data berbentuk array dengan nama LARIK
  typedef int LARIK[5];

  // Menggunakan tipe data LARIK
  // untuk mendeklarasikan variabel A
  LARIK A;
  int C; // Variabel bantu untuk melakukan pengulangan

  // Mengisikan nilai elemen ke dalam variabel A
  for (C=0; C<5; C++) {
    A[C] = (C+1) * 100;
  }
  // Menampilkan nilai elemen yang terdapat pada variabel A
  for (C=0; C<5; C++) {
    cout<<"A["<<C<<"] = "<<A[C]<<endl;
  }
  return 0;
}
```

Program di atas akan memberikan hasil seperti berikut.

```
A [0] = 100
A [1] = 200
A [2] = 300
A [3] = 400
A [4] = 500
```

7.9. Array dari Karakter

Dalam C++, kumpulan karakter tersebut dengan string (teks). Dan untuk mendeklarasikan array dari tipe berkarakter kita tentu akan menuliskannya ke dalam bentuk umum seperti ini.

```
char nama_Array [jumlah elemen];
```

Dengan demikian, apabila kita ingin melakukan deklarasi variabel string (misalnya dengan nama TEKS) yang terdiri dari 5 buah karakter, maka sintaksnya adalah sebagai berikut:

```
char TEKS[5] = {'B', 'U', 'D', 'I','\0'};
```

Karakter terakhir, '\0', disebut dengan karakter ***null***, yaitu karakter yang digunakan sebagai *terminator* dari sebuah string di dalam C++. Namun karena bentuk tersebut susah untuk ditulis dan riskan terhadap terjadinya sebuah kesalahan, maka C++ memperbolehkan kitsa untuk dapat menuliskan karakter-karakter tersebut dalam sebuah string yaitu dengan cara seperti di bawah ini:

```
char TEKS[5] = "BUDI";
```

Ukuran dari tipe data char adalah 1 byte sehingga ukuran memori yang dibutuhkan untuk proses pendeklarasian array di atas adalah 5 byte, yaitu 4 byte untuk *string* "BUDI" dan 1 byte untuk karakter *null.*

7.10 Array dari Struktur dab Struktur Array

Bagi kebanyakan programmer pemula terkadang istilah array dari struktur (*array of structure*) dan struktur dari array (*structure of array*) ini agak sedikit membingungkan. Sebenarnya konsepnya sederhana, array dari struktur berarti kita mendeklarasikan sebagai array dimana nilai dari setiap elemennya bertipe struktur. Sedangkan struktur dari array berarti kita mendeklarasikan sebuah

struktur dimana anggota dari struktur tersebut bertipe array. Untuk lebih memperjelas, berikut ini program yang di dalamnya terdapat sebuah array yang elemen-elemennya bertipe struktur.

```
#include <iostream>

using namespace std;

int main() {

 int C;// Mendeklarasikan variabel C
       // sebagai indeks pengulangan

 // Mendefinisikan tipe data bertipe struktur
 typedef struct SISWA {
   long NIM;
   char Nama[25];
   char Kota[30];
 };

 // Mendeklarasikan array A dengan tipe SISWA
 // dan jumlah elemennya tiga
 SISWA A[3];

 // Mengisikan nilai pada elemen array
 for (C=0; C<3; C++) {
  cout<<"NIM        : "; cin>>A[C].NIM;
  cout<<"Nama       : "; cin>>A[C].Nama;
  cout<<"Kota       : "; cin>>A[C].Kota;
  cout<<'\n';
 }

 // Menampilkan nilai yang telah dimasukkan
 // ke dalam elemen array
 for (C=0; C<3; C++) {
  cout<<A[C].NIM<<endl;
  cout<<A[C].Nama<<endl;
  cout<<A[C].Kota<<endl;
  cout<<'\n';
 }
 return 0;
}
```

Hasil yang akan diperoleh dari sintaks program diatas adalah sebagai berikut:

```
NIM         : 1
NAMA        : BUDI
KOTA        : BANDUNG
NIM         : 2
NAMA        : ARISTA
KOTA        : CIREBON
NIM         : 3
NAMA        : LINDA
KOTA        : BOGOR
 1
 BUDI
 BANDUNG
 2
 ARISTA
 CIREBON
 3
 LINDA
 BOGOR
```

Setiap elemen array A bertipe struktur yang mempunyai tiga anggota yaitu NIM, NAMA, dan KOTA. Hal ini menyebabkan pada saat kita mengakses setiap elemen array tersebut, maka data yang diminta juga ada tiga. Mengenai struktur dari array (*structure of array*),perhatikan program di bawah ini.

```
#include <iostream>
using namespace std;
int main() {
 // Mendeklarasikan struktur yang data anggotanya bertipe array
 struct STRUKTUR {
    int A[3]; int B[3];   };
 // Mendeklarasikan variabel X yang bertipe STRUKTUR
 STRUKTUR X;
 int C;
 // Memasukkan nilai ke dalam variabel X
 for (C=0; C<3; C++) {
    X.A[C] = C+1;
    X.B[C] = (C+1) * 100;  }
 // Menampilkan nilai yang telah dimasukkan ke dalam variabel X
 for (C=0; C<3; C++) {
   cout<<"X.A["<<C<<"] = "<<X.A[C]<<endl;
   cout<<"X.B["<<C<<"] = "<<X.B[C]<<endl;
   cout<<'\n';  }
 return 0;
}
```

Hasil yang akan diberikan dari program di atas adalah sebagai berikut:

```
X.
A [0] = 1
X.B [0] = 100
A [1] = 2
X.B [1] =200
A [2] = 3
X.B [2] =300
```

7.11 Array Multidimensi

Array multidimensi yaitu array yang terdiri dari beberapa subskrip array. Sebagai contoh, array 2 dimensi adalah array yang mempunyai 2 subskrip array, 3 dimensi mempunyai 3 subskrip array dan seterusnya. Array seperti ini sering digunakan untuk pemrosesan matrik. Pada bagian inikita akan mempelajari bagaimana C++ dapat memproses sebuah array yang terdiri dari dua subskrip array atau lebih.

7.12 Array Dua Dimensi

Seperti yang telah dikemukakan sebelumnya bahwa array dua dimensi adalah array yang mempunyai dua buah subskrip, yaitu baris dan kolom. Bentuk umum pendeklarasian sebuah dimensi di dalam C++ adalah sebagai berikut:

```
tipe _data
nama_array [jumlah_elemen_baris] [jumlah_elemen_kolom]
```

Misalnya, jika kita akan melakukan penjumlahan 2 buah matrik ordo 3x2, maka contoh sintaks program yang akan kita tuliskan adalah seperti yang terlihat di bawah ini.

```
#include <iostream>

using namespace std;

int main() {

  // Mendefinisikan tipe data yang berbentuk array dua dimensi
  typedef int MATRIK32 [3][2];

  // Mendeklarasikan array A sebagai array dua dimensi
  MATRIK32 A, B, C;

  int j, k;           // Mendeklarasikan variabel
              // untuk indeks pengulangan

  // Mengisikan nilai ke dalam elemen-elemen array A
  for (j=0; j<3; j++) {
    for (k=0; k<2; k++) {
      cout<<"A["<<j<<"]["<<k<<"] = "; cin>>A[j][k];
    }
  }
  cout<<endl;

  // Mengisikan nilai ke dalam elemen-elemen array B
  for (j=0; j<3; j++) {
    for (k=0; k<2; k++) {
      cout<<"B["<<j<<"]["<<k<<"] = "; cin>>B[j][k];
    }
  }
  cout<<endl;

  // Melakukan penjumlahan A dan B
  // dan menyimpan hasilnya ke dalam array C
  for (j=0; j<3; j++) {
    for (k=0; k<2; k++) {
      C[j][k] = A[j][k] + B[j][k];
    }
  }

  // Menampilkan hasil penjumlahan
  for (j=0; j<3; j++) {
    for (k=0; k<2; k++) {
      cout<<"C["<<j<<"]["<<k<<"] = "<<C[j][k]<<endl;
    }
  }

  return 0;
}
```

Contoh hasil yang akan diberikan dari program di atas adalah sebagai berikut:

```
A [0] [0] = 1
A [0] [1] = 2
A [1] [0] = 3
A [1] [1] = 4
A [2] [0] = 5
A [2] [1] = 6

B [0] [0] = 1
B [0] [1] = 2
B [1] [0] = 3
B [1] [1] = 4
B [2] [0] = 5
B [2] [1] = 6

C [0] [0] = 1
C [0] [1] = 2
C [1] [0] = 3
C [1] [1] = 4
C [2] [0] = 5
C [2] [1] = 6
```

Secara matematis, contoh hasil dari program di atas dapat dituliskan seperti di bawah ini:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} + \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} = \begin{bmatrix} 2 & 4 \\ 6 & 8 \\ 10 & 12 \end{bmatrix}$$

7.13 Inisialisasi pada Array Multidimensi

Sama seperti array satu dimensi, pada array multidimensi juga dapat dilakukan inisialisasi nilai dalama elemen-elemennya. Adapun cara melakukannya adalah seperti contoh yang terlihat di bawah ini.

```
int A[3] [3] = {1,2,3,4,5,6,7,8,9};
```

Namun untuk memudahkan proses inisialisasi, C++ mengizinkan kita untuk melakukan penelompokan untuk setiap baris yaitu dengan sintaks seperti berikut

```
int A [3] [3] = {(1,2,3), (4.5.6), (7,8,9)};
```

Untuk membuktikan hal tersebut, perhatikan contoh program di bawah ini

```
#include <iostream>

using namespace std;

int main() {

  // Melakukan inisialisasi nilai
  // ke dalam elemen-elemen array dua dimensi
  int A[3][3] = { {1,2,3}, {4,5,6}, {7,8,9} };

  // Mendeklarasikan variabel untuk indeks pengulangan
  int j, k;

  // Menampilkan nilai yang tersimpan dalam elemen array
  for (j=0; j<3; j++) {
    for (k=0; k<3; k++) {
      cout<<"A["<<j<<"]["<<k<<"] = "<<A[j][k]<<endl;
    }
    cout<<endl;
  }
  return 0;
}
```

Hasil yang akan diberikan dari program di atas adalah sebagai berikut:

```
A [0] [0] = 1
A [0] [1] = 2
A [0] [2] = 3

A [1] [0] = 4
A [1] [1] = 5
A [1] [2] = 6

A [2] [0] = 7
A [2] [1] = 8
A [2] [2] = 9
```

Secara matematis, array di atas sebenarnya adalah sebuah matrik dengan ordo 3x3, yang dapat dituliskan seperti di bawah ini.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

BAB 8
FUNGSI

Dalam C++, program merupakan kumpulan dari fungsi-fungsi, baik itu yang didefinisikan langsung dalam program maupun yang disimpan adalah suatu file header. C/C++ sendiri mempunyai fungsi utama yang disebut fungsi **main ().** Fungsi main() ini selalu ada dalam setiap program C/C++ dan compiler akan menjalankan program melalui perintah-perintah yang terdapat dalam fungsi ini.

Fungsi merupakan subprogram dan berguna untuk menjadikan program dapat lebih bersifat modular sehingga akan mudah dipahami dan dapat digunakan kembali, baik untuk program itu sendiri maupun untuk program lain yang memiliki proses yang sama. Dalam C++, fungsi dibedakan menjadi dua, yaitu ***user defined function*** dan ***built in function***. *User- defined function* adalah fungsi-fungsi yang didefinisikan sendiri , sedangkan *built in function* adalah fungsi-fungsi 'siap pakai' yang telah disediakan oleh compiler.

8.1 Fungsi Tanpa Nilai Balik

C++ tidak mengenal istilah prosedur seperti pada saat kita melakukan pemrogrman dengan menggunakan bahasa PASCAL. Dalam bahasa PASCAL sendiri prosedur didefinisikan sebagai suatu proses yang tidak mengembalikan nilai. Untuk melakukan hal ini di dalam C++, maka kita harus membuat suatu fungsi dengan tipe **void**, yang berarti tidak memiliki nilai balik (*return value*). Adapun bentuk umum dari pembuatan fungsi tanpa nilai balik ini adalah seperti yang terlihat di bawah ini.

```
void nama_fungsi (parameter, parameter2, . . .) {
statemen_yang_akan_dilakukan;
……
}
```

Bentuk umum untuk pemanggilan sebuah fungsi yang sebelumnya telah didefinisikan adalah seperti berikut.

```
nama_fungsi (nilai_parameter1, nilai_parameter2, . . .) ;
```

Sebagai contoh penggunaannya, di sini kita akan membuat sebuah fungsi yang dapat menuliskan teks "Aku sangat menyukai C++" sebanyak sepuluh kali, Adapun sintaks programnya adalah sebagai berikut:

```
#include <iostream>

using namespace std;

// Membuat fungsi dengan nama Tulis10Kali
void Tulis10Kali() {
  for (int C=0; C<10; C++) {
    cout<<"Aku sangat menyukai C++"<<endl;
  }
}

// Fungsi utama dalam program C++
int main() {

  // Memanggil fungsi Tulis10Kali untuk dieksekusi
  Tulis10Kali();

  return 0;
}
```

Hasil yang akan diberikan dari program di atas adalah sebagai berikut:

```
Aku sangat menyukai C++
Aku sangat menyukai C++
Aku sangat menyukai C++
Aku sangat menyukai C++
Aku sangat menyukai C++
Aku sangat menyukai C++
Aku sangat menyukai C++
Aku sangat menyukai C++
Aku sangat menyukai C++
Aku sangat menyukai C++
```

8.2. Fungsi dengan Nilai Balik

Fungsi ini berguna untuk melakukan suatu proses yang dapat mengembalikan sebuah nilai. Dalam membuat fungsi ini kita harus mendefinisikan tipe data dari nilai yang akan dikembalikan. Berikut ini adalah bentuk umum dari pembuatan fungsi yang mempunyai nilai balik.

```
tipe_data nama_fungsi (parameter1, parameter2,…){
statemen_yang_akan_dilakukan;
…..
return nilai_yang_akan_dikembalikan;
}
```

Sedangkan untuk pemanggilannya sama seperti pada fungsi yang tidak mempunyai nilai balik. Untuk dapat lebih memahami pembuatannya, perhatikan contoh-contoh program di bawah ini.

8.3 Fungsi yang Mengembalikan Tipe String

```
#include <iostream>

using namespace std;

// Membuat fungsi sederhana yang mengembalikan tipe string
char* TestFungsiString() {
  return "Ini adalah nilai dari fungsi";
}

// Fungsi utama
int main() {
  // Memanggil dan menampilkan hasil fungsi
  cout<<TestFungsiString();

  return 0;
}
```

Hasil yang akan diberikan dari program di atas adalah sebagai berikut:

```
Ini  adalah nilai dari fungsi
```

8.4 Fungsi yang mengembalikan Tipe Bilangan

```
#include <iostream>

using namespace std;

// Membuat fungsi dengan nilai kembalian bertipe double
double TestFungsiBilangan() {
  return (3.14 * 2);
}

// Fungsi utama
int main() {

  cout<<"Nilai yang terdapat dalam fungsi : ";
  cout<<TestFungsiBilangan();

  return 0;
}
```

Hasil yang akan diberikan oleh program di atas adalah sebagai berikut:

```
Nilai yang terdapat dalam fungsi : 6.28
```

8.5 Fungsi dengan Parameter

Secara teori, parameter yang terdapat pada pendefinisian sebuah fungsi disebut dengan *parameter formal,* sedangkan parameter yang terdapat pada saat pemanggilan disebut dengan *parameter aktual.* Jumlah serta tipe data antara parameter formal dan parameter aktual haruslah sesuai, jika tidak maka compiler akan menampilkan pesan kesalahan.

8.5.1 Jenis Parameter

Terdapat tiga buah jenis parameter yang dapat dilewatkan pada sebuah fungsi dalam C++, yaitu *parameter masukan, keluaran,* dan *masukan/keluaran.*

a. Parameter Masukan

Sesuai dengan namanya parameter ini akan digunakan sebagai nilai masukan dalam sebuah fungsi. Nilai tersebut kemudian akan diproses oleh fungsi untuk menghasilkan sebuah nilai kembalian (*return value*). Berikut ini contoh program yang di dalamnya terdapat sebuah fungsi dengan parameter yang bertipe masukan.

b. Parameter Keluaran

Parameter keluaran adalah parameter yang berfungsi untuk menampung nilai yang dihasilkan dari proses di dalam fungsi. Parameter keluaran pada umumnya digunakan di dalam fungsi void (fungsi yang tidak mengembalikan nilai). Dengan kata lain, parameter tersebut berguna sebagai nilai keluaran dari sebuah fungsi. Maka dari itu, parameter keluaran ini harus dilewatkan berdasarkan alamat atau referensinya, yaitu dengan menggunakan pointer atau *reference*.

c. Parameter Masukan/ Keluaran

Jenis parameter ini adalah parameter yang digunakan sebagai masukan dan juga keluaran. Artinya mula-mula nilai dari parameter ini akan digunakan sebagai masukan untuk proses di dalam fungsi, setelah proses selesai maka hasilnya akan disimpan kembali ke dalam parameter tersebut sebagai nilai keluaran.

8.5.2 Melewatkan Parameter Berdasarkan Nilai (*Pass by Value*)

Terdapat dua buah cara untuk melewatkan parameter di dalam sebuah fungsi, yaitu berdasarkan nilai (*pass by value*) dan berdasarkan alamat (*pass by reference*) . *Pass by value*, sesuai dengan namanya, fungsi ini akan melewatkan nilai parameter ke dalam sebuah fungsi untuk digunakan sesuai proses yang terdapat di dalam fungsi tersebut. Jika kita melewatkan parameter dengan cara ini maka

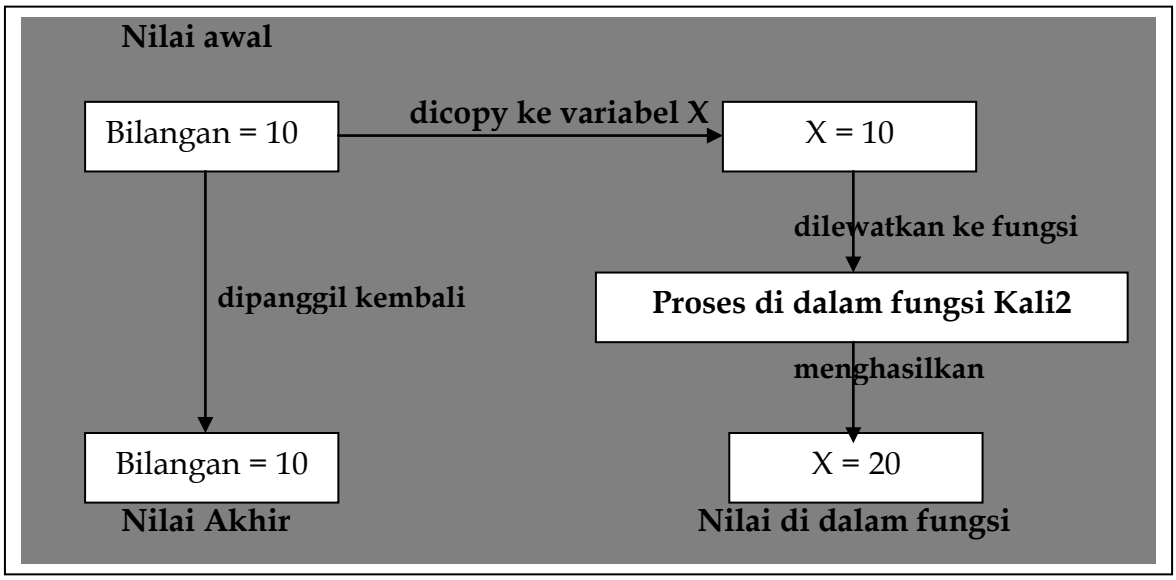
nilai yang dihasilkan oleh fungsi tidak akan mempengaruhi nilai yang terdapat pada program (di luar fungsi tersebut). Hal ini disebabkan karena pada saat pemanggilan fungsi, compiler hanya akan membuat salinan (*copy*) dari nilai yang terdapat pada parameter actual ke parameter formal. Dengan kata lain, yang akan berubah adalah nilai di dalam fungsi saja.

```
#include <iostream>

using namespace std;

// Membuat fungsi dengan melewatkan nilai X ke dalamnya
void Kali2(int X) {
  X = X * 2;
  cout<<"Nilai di dalam fungsi : "<<X<<endl;
}

// Fungsi utama
int main() {
  int Bilangan;

  cout<<"Masukkan sebuah bilangan bulat : ";
  cin>>Bilangan;
  cout<<endl;

  // Menampilkan nilai awal
  cout<<"Nilai awal : "<<Bilangan<<endl;

  // Memanggil fungsi Kali2
  Kali2(Bilangan);

  // Menampilkan nilai akhir
  cout<<"Nilai awal : "<<Bilangan<<endl;

  return 0;
}
```

Contoh hasil yang akan diberikan dari program di atas adalah sebagai berikut:

```
Masukkan sebuah bilangan bulat : 10
Nilai awal                     : 10
Nilai di dalam fungsi          : 20
Nilai akhir                    : 10
```

Jika kita amati hasil program di atas, maka kasus tersebut dapat diilustrasikan dengan gambar di bawah ini:



Pada kasus di atas, variabel Bilangan dan variabel X menempati alamat memori yang berbeda sehingga pada saat pemanggilan fungsi, nilai dari keduanya tentu akan berbeda. Di sini, yang berubah hanya nilai darI variabel X, sedangkan variabel Bilangan bernilai tetap karena kita memang tidak melakukan apa-pa terhadap variabel tersebut.

8.5.3 Melewatkan Parameter Berdasarkan Alamat (*Pass by Reference)*

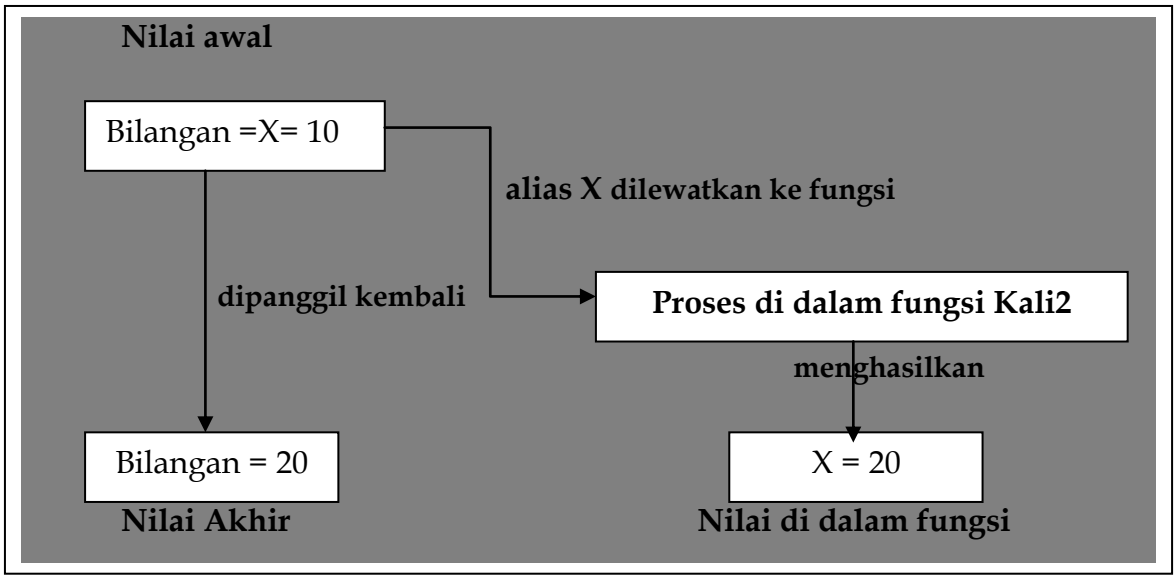
Pass by Reference adalah melewatkan parameter ke sebuah fungsi berdasarkan alamatnya. Melewatkan parameter dengan cara ini akan menyebabkan nilai yang terdapat di dalam fungsi akan sama persis dengan nilai yang terdapat pada program (di luar fungsi). Hal ini disebabkan karena adanya alamat dari variabel yang berperan sebagai parameter formal sama dengan alamat dari variabel yang berperan sebagai parameter aktual. Berikut ini adalah contoh program pengiriman parameter dengan berdasarkan alamat

```
#include <iostream>
using namespace std;

// Mendefinsikan fungsi yang melewatkan parameternya
// berdasarkan alamat
void Kali2(int& X) {  // Gunakan tanda &
                      // untuk membuat alias atau reference
  X = X * 2;
  cout<<"Nilai di dalam fungsi : "<<X<<endl;
}
int main() {
  int Bilangan;
  cout<<"Masukkan sebuah bilangan bulat : "; cin>>Bilangan;
  cout<<endl;

  // Menampilkan nilai awal
  cout<<Bilangan<<endl;
  Kali2(Bilangan);     // Memanggil fungsi Kali2

  // Menampilkan nilai akhir
  cout<<Bilangan<<endl;

  return 0;
}
```

Hasil dari program diatas adalah sebagai berikut

```
Masukkan sebuah bilangan bulat : 10
Nilai awal     :10
Nilai di dalam fungsi : 20
Niai akhir : 20
```

Dari hasil diatas dapat disimpulkan, bahwa X bukanlah sebuah variabel melainkan sebuah alias (nama lain) atau *reference* dari variabel Bilangan. Maka dari itu, alias X dan variabel Bilangan tentu memiliki alamat yang sama karena keduanya sebenarnya merupakan objek yang sama. Dengan demikian, jika nilai X berubah, maka nilai yang terdapat pada variabel Bilangan pun akan ikut berubah. Jika kita amati hasil program di atas, maka kasus tersebut dapat diilustrasikan dengan gambar di bawah ini:



Berbeda dengan *pass by value*, disini nilai X dan nilai yang terdapat pada variabel Bilangan akan saling mempengaruhi.

8.5.4 Melewatkan Parameter Bertipe Array

Dalam C++, array juga dapat digunakan sebagai parameter dalam sebuah fungsi. Namun parameter array ini sering diganti dengan parameter yang bertipe pointer.

contoh program dapat dilihat dibawah ini

```
#include <iostream>

using namespace std;

// Mendefinisikan fungsi untuk proses input array
void InputArray(int A[] , int N) {
  for (int C=0; C<N; C++) {
    cout<<"Masukkan nilai A["<<C<<"] : "; cin>>A[C];
  }
}

// Mendefinisikan fungsi untuk menghitung jumlah (sum)
// dari semua elemen array
long Jumlah(int A[], int N) {
  long jml = 0;
  // Menjumlahkan semua elemen array
  for (int C=0; C<N; C++) {
    jml += A[C];
  }
  return jml;
}

// Fungsi utama
int main() {
  int X[100];   // Elemen maksimal adalah 100
  int BanyakElemen;
  long HASIL;

  cout<<"Masukkan banyaknya elemen yang diinginkan : ";
  cin>>BanyakElemen;
  cout<<endl;

  // Memanggil fungsi InputArray
  InputArray(X, BanyakElemen);

  // Memanggil fungsi Jumlah dan menampung hasilnya
  // ke variabel HASIL
  HASIL = Jumlah(X, BanyakElemen);

  // Menampilkasn hasil
  cout<<"\nHasilnya = "<<HASIL;

  retun 0;
}
```

Contoh hasil yang akan diberikan dari program diatas adalah sebagai berikut:

```
Masukkan banyakanya elemen array yang diinginkan : 5
Masukkan niali A[0] : 10
Masukkan niali A[1] : 20
Masukkan niali A[2] : 30
Masukkan niali A[3] : 40
Masukkan niali A[4] : 50

Hasilnya = 150
```

8.5.5 Melewatkan Parameter yang Bersifat Konstan

Untuk menggunakan parameter ini hanya menambahkan kata kunci **const** di depan parameter tersebut. Untuk lebih jelasnya kita perhatikan contoh program berikut ini.

```
#include <iostream>

using namespace std;

// Mendefinisikan fungsi untuk menghitung keliling lingkaran
double KelilingLingkaran(const float PI, int jari_jari) {
  return (2*PI*jari_jari);
}

// Fungsi utama
int main() {
  int R;
  double HASIL;

  cout<<"Masukkan panjang jari-jari lingkaran : "; cin>>R;

  // Memanggil fungsi KelilingLingkaran
  HASIL = KelilingLingkaran(3.14, R);

  // Menampilkan hasil yang didapatkan
  cout<<"Kelilingnya = "<<HASIL;

  return 0;
}
```

Contoh hasil yang akan diberikan dari program di atas adalah sebagai berikut:

```
Masukkan panjang jari-jari lingkaran: 1
Kelilingnya = 6.28
```

8.5.6 Mengeset Parameter dengan Nilai *Default*

Nilai dari parameter yang terdapat dalam sebuah fungsi dapat kita set dengan nilai *default,* artinya jika kita tidak mendefinisikannya pada saat pemanggilan, maka nilai yang akan digunakan fungsi adalah nilai *default*. Untuk melakukan hal ini kita hanya mengisikan nilai parameter bersangkutan pada saat kita mendefinisikan fungsi. Bentuk umum dari pembuatan fungsi yang menggunakan nilai *default* adalah seperti yang tampak di bawah ini.

```
tipe_data nama_fungsi (tipe_data parameter =  nilai_default, . . .)
{
……
 return nilai_kembalian;
}
```

Berikut ini contoh program yang di dalamnya terdapat sebuah

```
#include <iostream>
using namespace std;
// Mendefinisikan fungsi untuk menghitung volume balok
long VolumeBalok(int panjang, int lebar = 20, int tinggi = 5) {
  return (panjang * lebar * tinggi);
}
int main() {
  int P = 50;     // Mendeklarasikan variabel P dengan nilai 50
  int L = 10;     // Mendeklarasikan variabel L dengan nilai 10
  int T = 2;      // Mendeklarasikan variabel T dengan nilai 2
  long HASIL;
  // Memanggil fungsi dengan tiga parameter
  HASIL = VolumeBalok(P, L, T);
  cout<<"Volume Balok = "<<HASIL<endl;
  // Memanggil fungsi dengan dua parameter
  HASIL = VolumeBalok(P, L);
  cout<<"Volume Balok = "<<HASIL<endl;
  // Memanggil fungsi dengan satu parameter
  HASIL = VolumeBalok(P);
  cout<<"Volume Balok = "<<HASIL<endl;
  return 0;
}
```

Hasil yang akan diberikan dari program diatas adalah sebagai berikut:

```
Volume Balok = 1000
Volume Balok = 1000
Volume Balok = 1000
```

8.6 Pointer Ke Fungsi

Meskipun fungsi bukan sebuah variabel, namun fungsi masih merupakan objek yang memiliki lokasi fisik di memori. Hal ini menunjukkan bahwa nilai dari sebuah fungsi dapat kita ambil melalui pointer. Contoh sintaks yang dapat dituliskan adalah sebagai berikut:

```
#include <iostream>
using namespace std;

// Mendefinsikan fungsi tambah
int Tambah(int X, int Y) {
  return (X + Y);
}
// Fungsi utama
int main() {
  // Mendeklarasikan pointer ke fungsi Tambah() dengan parameter (int, int)
  int (*P) (int, int);

  int HASIL;          // Variabel untuk menampung nilai balik
  int a=20, b =5;     // Variabel yang akan digunakan sebagai parameter

  // Memanggil fungsi tambah dan menyimpan nilainya
  // ke variabel HASIL
  HASIL = Tambah(a, b);
  // Memerintahkan P untuk menunjuk alamat dari fungsi Tambah()
  P = Tambah;
  // Menampilkan hasil melalui pointer P
  cout<<"Nilai (*P)(20,5)     : "<<(*P)(a,b)<<endl;
  // Menampilkan hasil melalui variabel HASIL
  cout<<"Nilai HASIL                  : "<<HASIL<<endl;
  // Menampilkan alamat yang ditunjuk oleh pointer P
  cout<<"Nilai P                : "<<P<<endl;
  // Menampilkan alamat dari fungsi Tambah
  cout<<"Nilai Tambah       : "<<Tambah<<endl;

  return 0;
}
```

Contoh hasil yang akan diberikan dari program di atas adalah sebagai berikut:

```
Nilai (*P) (20,5)      : 25
Nilai HASIL            : 25
Nilai P                : 00401150
Nilai Tambah           : 00401150
```

8.7 Penimpaan Fungsi (*Function Overloading*)

Salah satu kelebihan C++ dibandingkan bahasa C adalah karena C++ mendukung fungsi *overload*. Fungsi *overload* didefinisikan sebagai fungsi-fungsi dengan nama yang sama tetapi memiliki parameter berbeda. Kata 'berbeda' di sini dapat dilihat dari tiga esgi, yaitu dari segi jumlah, tipe data, dan gabungan dari keduanya. Dalam fungsi *overload,* pada saat kita melakukan pemanggilan fungsi, kompiler akan memilih fungsi yang parameter aktualnya sesuai dengan parameter formalnya. Berikut ini contoh program yang menunjukkan penggunaan fungsi *overload.*

a. Fungsi overload dengan jumlah parameter berbeda

```
#include <iostream>
using namespace std;

// Mendefinisikan fungsi Tulis dengan satu parameter
void Tulis(char* S) {
  cout<<S<<endl;
}
// Mendefinisikan fungsi Tulis dengan dua parameter
void Tulis(char* S1, char* S2) {
  cout<<S1<<" "<<S2<<endl;
}
// Mendefinisikan fungsi Tulis dengan tiga parameter
void Tulis(char* S1, char* S2, char* S3) {
  cout<<S1<<" "<<S2<<" "<<S3<<endl;
}
// Fungsi utama
int main() {
  // Melakukan pemanggilan fungsi Tulis
  Tulis("Budi");
  Tulis("Penerbit", "INFORMATIKA");
  Tulis("Mengungkap", "Rahasia", "C++");
  return 0;
}
```

Hasil yang akan diberikan dari program di atas adalah sebagai berikut:

```
Budi
Penerbit INFORMATIKA
Mengungkap Rahasia C++
```

b. Fungsi overload dengan tipe data parameter berbeda

```
#include <iostream>

using namespace std;

// Mendefinisikan fungsi Tulis dengan parameter bertipe char*
void Tulis(char* X) {
  cout<<X<<endl;
}

// Mendefinisikan fungsi Tulis dengan parameter bertipe int
void Tulis(int X) {
  cout<<X<<endl;
}

// Mendefinisikan fungsi Tulis dengan parameter bertipe double
void Tulis(double X) {
  cout<<X<<endl;
}

// Fungsi utama
int main() {

  // Melakukan pemanggilan fungsi Tulis
  Tulis("C++");
  Tulis(100);
  Tulis(21.0378);

  return 0;
}
```

Hasil yang akan diberikan dari program di atas adalah sebagai berikut:

```
C++
100
21.0378
```

c. Fungsi *overload* dengan jumlah dan tipe data parameter berbeda

```
#include <iostream>

using namespace std;

// Mendefinisikan fungsi Tulis dengan parameter berjumlah 1
// dan bertipe int
void Tulis(int X) {
  cout<<X<<endl;
}

// Mendefinisikan fungsi Tulis dengan parameter berjumlah 2
// dan bertipe char*
void Tulis(char* S1, char* S2) {
  cout<<S1<<" "<<S2<<endl;
}

// Fungsi utama
int main() {

  // Melakukan pemanggilan fungsi Tulis
  Tulis(100);
  Tulis("Penerbit", "INFORMATIKA");

  return 0;
}
```

Hasil yang akan diberikan dari program di atas adalah sebagai berikut:

```
100
Penerbit INFORMATIKA
```

8.8 Mengembalikan Nilai Bertipe Pointer

Kita dapat membuat fungsi yang mengembalikan nilai berupa pointer. Untuk melakukan hal ini tentu kita harus mendeklarasikan tipe data dari nilai kembalian dengan pointer. Berikut ini contoh program yang di dalamnya mengandung fungsi yang mengembalikan pointer ke tipe **char**.

```
#include <iostream>

using namespace std;

// Mendefinisikan fungsi untuk mengembalikan string
// dari karakter yang dicari
char *CariKarakter(char K, char *S) {
  // Selama K tidak sama dengan *S dan *S masih ada
  while ((K != *S) && (*S)) {
    S++;                          // Cari di alamat berikutnya
  }
  return S;
}

// Fungsi utama
int main() {
  char *P;       // Pointer untuk menampung nilai kembalian fungsi
  char* STRING = "Mengungkap Rahasia C++";
  char Karakter = 'C';

  // Memanggil fungsi CariKarakter
  P = CariKarakter(Karakter, STRING);

  if (*P)                  // Jika ditemukan
     cout<<"Karakter "<<Karakter<<" ditemukan pada "<<P;
  else             // Jika tidak ditemukan
     cout<<"Karakter "<<Karakter<<" tidak ditemukan";

  return 0;
}
```

Hasil yang akan diberikan dari program di atas adalah sebagai berikut:

```
Karakter C ditemukan pada C++
```

8.9 Membuat *Prototype* Fungsi

Dalam C++, fungsi-fungsi dapat dideklarasikan terlebih dahulu sebelum dilakukan pendefinisian. Definisi fungsi baru dibuat setelah pembuatan fungsi utama. Hai ini disebut dengan istilah *prototype*. *Prototype* akan mempermudah kita dalam mengenali daftar fungsi yang tersedia atau akan didefinisikan di dalam

program. Adapun bentuk umum dari pembuatan *prototype* fungsi adalah seperti tampak di bawah ini.

```
tipe_data nama_fungsi (parameter1, parameter2, ...);
```

Berikut ini contoh program yang di dalamnya terdapat *prototype* fungsi.

```
#include <iostream>

using namespace std;

// Membuat prototype (pendeklarasian fungsi)
int Kali(int X, int Y);// Mendeklarasikan fungsi Kali
void Tulis(int S);                // Mendeklarasikan fungsi Tulis

// Fungsi Utama
int main() {
  int Bilangan1, Bilangan2, HASIL;

  cout<<"Masukkan bilangan pertama : "; cin>>Bilangan1;
  cout<<"Masukkan bilangan kedua   : "; cin>>Bilangan2;
  cout<<endl;

  // Menggunakan fungsi Kali
  HASIL = Kali(Bilangan1, Bilangan2);

  // Menggunakan fungsi Tulis
  Tulis(HASIL);

  return 0;
}
```

Contoh hasil yang akan diberikan dari program di atas adalah sebagai berikut :

```
Masukkan bilangan pertama        : 10
Masukkan bilangan kedua          : 25

250
```

8.10 Fungsi Inline (*Inline Function*)

Jika kita melakukan pemanggilan fungsi sebanyak 5 kali, maka kompiler juga akan melakukan proses tersebut sebanyak 5 kali. Apabila fungsi yang kita definisikan hanya terdiri dari sedikit statemen (misalnya satu atau dua baris), hal ini tentu akan mengurangi efisiensi program karena kompiler harus meloncat keluar masuk untuk melakukan proses tersebut. Kata 'efisiensi' bagi seorang programmer tentu akan dihubungkan dengan masalah 'kecepatan' eksekusi sebuah program. Untuk menghindari hal ini, C++ menyediakan fitur yang disebut dengan fungsi *inline* (*inline function*), yaitu dengan menggunakan kata kunci **inline.**

Konsep dasar dari *inline function* adalah proses penyalinan (*copy*) baris yang terdapat pada definisi fungsi ke baris pada saat kita melakukan pemanggilan fungsi tersebut. Artinya, jika kita membuat fungsi *inline*, kompiler tidak menyimpannya ke dalam memori melainkan hanya akan membuat salinan kode dari fungsi tersebut. Hal ini tentu tidak membutuhkan proses peloncatan statemen seperti pada saat kita membuat fungsi biasa sehingga proses eksekusinya akan lebih cepat. Berikut ini program yang menunjukkan pembuatan fungsi *inline.*

```
#include <iostream>
using namespace std;

// Mendefinisikan fungis inline yang mengalikan bilangan dengan 2
inline int Kali2(int X) {
  return X * 2; }
// Fungsi utama
int main() {
  int HASIL;
  //Melakukan pemanggilan fungsi inline untuk pertama kali
  HASIL = Kali2(10);
  cout<<"Hasil = "<<HASIL<<endl;
  //Melakukan pemanggilan fungsi inline untuk kedua kali
  HASIL = Kali2(20);
  cout<<"Hasil = "<<HASIL<<endl;
  //Melakukan pemanggilan fungsi inline untuk ketiga kali
  HASIL = Kali2(30);
  cout<<"Hasil = "<<HASIL<<endl;
  return 0;
}
```

Hasil yang akan diberikan dari program di atas adalah sebagai berikut:

```
Hasil = 20
Hasil = 40
Hasil = 60
```

Jika kita amati cara kerjanya, sebenarnya kompiler akan menerjemahkan program di atas seperti sintaks berikut.

```
#include <iostream>

using namespace std;

// Mendefinisikan fungis inline yang mengalikan bilangan dengan 2
inline int Kali2(int X) {
  return X * 2;// yang dicetak tebal adalah statemen yang akan disalin (di-copy)
}

// Fungsi utama
int main() {
  int HASIL;

  //Melakukan pemanggilan fungsi inline untuk pertama kali
  HASIL = 10 * 2; // Melakukan penyalinan statemen untuk X = 10
  cout<<"Hasil = "<<HASIL<<endl;

  //Melakukan pemanggilan fungsi inline untuk kedua kali
  HASIL = 20 * 2; // Melakukan penyalinan statemen untuk X = 20
  cout<<"Hasil = "<<HASIL<<endl;

  //Melakukan pemanggilan fungsi inline untuk ketiga kali
  HASIL = 30 * 2; // Melakukan penyalinan statemen untuk X = 30
  cout<<"Hasil = "<<HASIL<<endl;

  return 0;
}
```

8.11 Rekursi

Rekursi adalah fungsi yang pada saat pendefinisiannya memanggil dirinya sendiri untuk melakukan proses di dalamnya. Contoh paling sederhana untuk menunjukkan proses rekursi adalah pada saat kita membuat program untuk

menghitung nilai faktorial dari sebuah bilangan bulat. Adapun sintaks programnya adalah seperti yang tampak di bawah ini.

```
#include <iostream>

using namespace std;

// Mendefinsikan fungsi Faktorial
int Faktorial(int X) {
  if (X==1) return(1);
  return X * Faktorial(X-1); // Memanggil dirinya sendiri
}

// Fungsi utama
int main() {
  int Bilangan, HASIL;
  cout<<"Masukkan bilangan yang akan dihitung : ";
  cin>>Bilangan;

  // Memanggil fungsi Faktorial
  HASIL = Faktorial(Bilangan);

  // Menampilkan hasil
  cout<<Bilangan<<"! = "<<HASIL;

  return 0;
}
```

Contoh hasil yang akan diberikan dari program di atas adalah sebagai berikut:

```
Masukkan bilangan yang akan dihitung : 5
5! = 120
```

Jika fungsi di atas dituliskan dengan fungsi non-rekursi, maka sintaksnya adalah sebagai berikut.

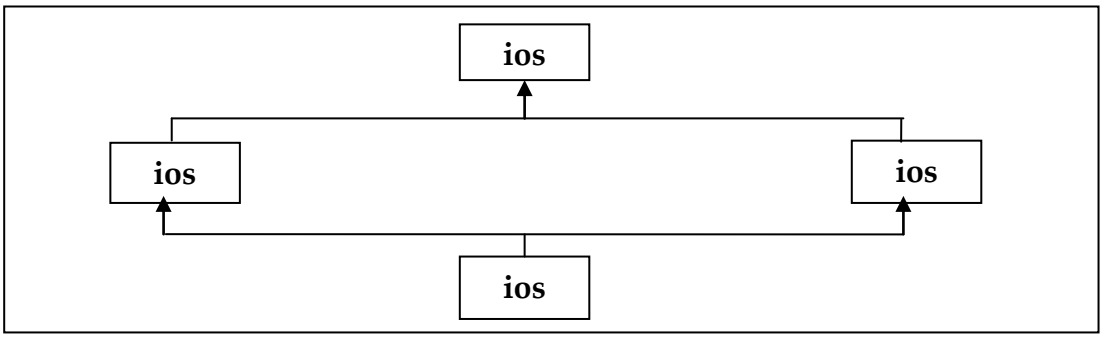
```
Int Faktorial (int x) {
        Hasil = 1;
        For (int C =1; C<=X; C++) {
        Hasil = hasil * C;
        }
        Return hasil;
}
```

BAB 9

INPUT/ OUTPUT DALAM C++

9.1 Pendahuluan

Sejauh ini kita telah banyak menggunakan **cout** (dibaca si *out*) untuk menuliskan ke layar dan **cin** (dibaca *si in*) untuk membaca nilai dari *keyboard* tanpa membahas lebih detil mengenai kegunaannya. Secara umum, hirarki kelas dalam C++ yang digunakan untuk proses input dan output adalah sebagai berikut:



Pada gambar di atas dapat kita lihat bahwa **ios** merupakan kelas dasar (*virtual base class*) yang berisi fasilitas untuk proses input dan output data. Di dalam kelas ini juga didefinisikan anggota-anggota yang dapat digunakan untuk proses pengesetan format data dalam proses input maupun output. Pembahasan mengenai pengesetan format ini akan kita ungkap lebih detil lagi pada sub bab selanjutnya di dalam buku ini. Kelas **ios** ini dijadikan sebagai kelas dasar dari kelas **istream** (singkatan dari *input stream*) dan **ostream** (singkatan dari *output stream*).

Kelas **istream** adalah kelas yang dibuat khusus untuk menangani masalah-masalah input dengan meng-ekstrak fasilitas-fasilitas input yang terdapat pada kelas **ios** dan tentunya juga melalui penambahan-penambahan yang lainnya. Sedangkan kelas ostream digunakan untuk menangani masalah-masalah output. Dari kedua kelas tersebut, kemudian dibuat lagi sebuah kelas baru yang dinamakan **iostream.** Oleh karena diturunkan dari dua buah indukm maka kelas iostream ini otomatis dapat menangani masalah-masalah yang bisa ditangani oleh kelas **istream** dan kelas **ostream.** Hal inilah yang menyebabkan kita menggunakan **iostream** sebagai standar untuk melakukan operasi input dan output (I/O) data.

9.2 Stream

Stream adalah suatu peralatan logika (*logical device*) yang berguna untuk mendapatkan atau memberikan informasi. Stream ini akan dihubungkan dengan peralatan fisik (seperti *keyboard, screen(*layar*)* maupun *printer*) melalui sistem I/ O. Semua stream mempunyai kelakuan yang sama, sehingga apabila suatu fungsu I/ O dapat dioperasikan ke peralatan fisik yang berbeda. Sebagai contoh, jjika kita kan melakukan penulisan data, maka cara yang digunakan untuk menuliskan ke layat maupun printer adalah sama. Dalam bahasa C, untuk melakukan hal-hal yang berhubungan dengan proses input dan output data digunakan library standar, yaitu **stdio**. Namun dalam C++, library standar yang digunakan adalam **iostream**.

Pada saat program C++ memulai proses ekskusi, terdapat empat buah stream yang secara otomatis akan terbuka, yaitu seperti yang terlihat pada tabel di bawah ini.

Nama Stream	Kegunaan	Peralatan Standar
cin	Input standar	*Keyboard*
cout	Output standar	Layar (*screen*)
cerr	Kesalahan output standar	Layar (*screen*)
clog	cerr yang ter-buffer melalui file *log*	Layar (*screen*)

Stream **cin**, **cout** dan **cerr** di dalam C++ ini berkoresponden dengan **stdin**, **stdou**t dan **stderr** di dalam bahasa C.

9.3 Input menggunakan *cin*

cin adalah suatu stream yang merespon proses input yang dilakukan. Stream ini hanya tersedia jika kita memasukkan file *header* **iostream.h** (untuk C++ lama) atau **<iostream>** (untuk C++ standar) di dalam program yag kita buat. Contoh program yang menunjukkan penggunaan stream cin.

```
#include <iostream>

using namespace std;

int main() {

 // Mendeklarasikan sebuah variabel yang bertipe int
 int X;

 // Memberikan informasi kepada user untuk melakukan input
 cout<<"Masukkan sebuah bilangan bulat : ";

 // Menggunakan cin untuk merespon input yang akan dilakukan
 cin>>X;

 // Menampilkan nilai X
 cout<<"Nilai X : "<<X;

 return 0;
}
```

Pada program di atas terdapat penggunaan stream **cout**, namun untuk sementara ini kita belum perlu mempedulikannya. Di sini, kita hanya akan fokus ke statemen yang melakukan input, yaitu **cin>>X**;. Statemen tersebut akan menyebabkan input yang dilakukan oleh user melalui *keyboard* akan disimpan ke dalam variabel **X**. Pada penggunaan stream cin digunakan pula operator **>>**. Operator ini telah di- overload sehingga dapat berguna untuk berbagai jenis parameter, seperti int, char, float dan sebagainya. Oleh karena di sini kita menggunakan variabel **X** yang bertipe int, maka fungsi yang akan dipanggil adalah sebagai berikut:

```
istream& operator>> (int &)
```

Hal inilah yang berarti jika kita memasukkan nilai **5** melalui *keyboard,* maka nilai tersebut akan disimpan ke dalam variabel X. Sebagai bukti, berikut ini contoh

```
Masukkan sebuah bikangan bulat      : 5
Nilai X                             : 5
```

Dengan menggunakan operator **>>**, kita dapat melakukan input dengan satu penulisan **cin**. Maksud dari pernyataan ini adalah kita dapat melakukan penyingkatan penulisan dalam proses input yang akan dilakukan. Sebagai contoh, terdapat sintaks seperti di bawah ini.

```
Int X, Y, Z;

cout<<"Masukkan nilai X: ";
cin>>X;

cout<<"Masukkan nilai Y: ";
cin>>Y;

cout<<"Masukkan nilai Z: ";
cin>>Z;
```

Maka sintaks tersebut dapat disingkat penulisannya dengan menggunakan sintaks berikut:

```
Int X, Y, Z;
cout<<"Masukkan nilai X, Y, dan Z: n \n";
cin>>X>>Y>>Z;
```

Hasil yang akan diberikan dari porgram di atas adalah sama. Artinya input pertama akan disimpan ke salam variabel **X**, input kedua akan disimpan ke dalam variabel **Y** dan input krtiga akan disimpan ke dalam variabel **Z**.

9.4. Output Menggunakan cout

Untuk melakukan output ke peralatan standar, yaitu layar (*screen*) adalah dengan cara menggunakan stream **cout**. Adapun operator yang digunakan adalah operator **<<.** Operator ini juga telah di *overload* sehingga dapat digunakan untuk berbagai macam tipe data. Berikut ini contoh penggunaan **cout** pada variabel tunggal dalam sebuah program.

```
#include <iostream>

using namespace std;

int main() {

  int X = 26;

  // Melakukan output terhadap nilai X
  cout<<X;

  return 0;
}
```

Sama seperti cin, cout juga dapat melakukan output secara beruntun. Artinya, satu penulisan cout dapat diikuti oleh banyak operator **<<**. Berikut ini contoh program yang menunjukkan hal tersebut.

```
#include <iostream>

using namespace std;

int main() {
  int X = 100, Y = 200, Z = 300;

  // Melakukan output terhadap nilai X, Y dan Z
  cout<<X<<' '<<Y<<' '<<Z;

  return 0;
}
```

9.5 Mengatur Format Input/ Output

C++ mengizinkan kita untuk mengeset format dari operasi-operasi I/ O yang dilakukan. Sebagai contoh, kita dapat mengeset lebar kolom dari data yang akan kita tampilkan ke layar, menentukan berapa digit angka di belakang koma dan yang lainnya. Dalam C++, terdapat cara untuk mengeset operasi I/ O, yaitu menggunakan fungsi-fungsi khusus yang disebut dengan *manipulator* yang dimasukkan sebagai bagian dari ekspresi I/ O.

9.6 Menggunakan *Manipulator*

Dalam C++, terdapat beberapa *manipulator* yang merupakan fitur baru yang ditambahkan. Hal ini berarti bahwa kompiler C++ lama tidak mendukung adanya *manipulator*. Adapun manipulator yang dimaksud disini adalah seperti yang terlihat pada tabel di bawah ini.

Manipulator	Kegunaan	Operasi
Boolalpha	Mengaktifkan flag boolalpha	Input/ output
Dec	Mengaktifkan flag dec	Input/ output
End1	Menampilkan baris baru dan membuang stream	Output
Ends	Menampilkan null	Output
Fixed	Mengaktifkan flag fixed	Output
Flush	Membuang stream	Output
Hex	Mengaktifkan flag hex	Input/ output
Internal	Mengaktifkan flag internal	Output
Left	Mengaktifkan flag left	Output
Noboolalpha	Mengaktifkan flag boolalpha	Input/ output
Noshowbase	Mematikan flag showbase	Output
Noshowpos	Mematikan flag showpos	Output
Noskipws	Mematikan flag skipws	Input
Nounitbuf	Mematikan flag unitbuf	Output
Nouppercase	Mematikan flag uppercase	Output
Oct	Mengaktifkan flag oct	Input/ output
Resetiosflags (fmtflags f)	Mematikan flag yang ditulis (f)	Input/ output
Right	Mengaktifkan flag right	Output
scientific	Mengaktifkan flag scientific	Output
Setbase (int *base*)	Mengeset nomor basis ke base	Input/ output
Setfill (int ch)	Mengisi karakter dengan ch	Output

Setiosflags (fmtflags f)	Mengaktifkan flag yang dituliskan (f)	Input/ output
Setprecision (int p)	Mengeset presisi digit	Output
Setw (int w)	Mengeset lebar kolom ke w	Output
Showbase	Mengaktifkan flag showbase	Output
Showpoint	Mengaktifkan flag showpoint	Output
Showpos	Mengaktifkan flag showpos	Output
Skipws	Mengaktifkan flag skipws	Input
Unitbuf	Mengaktifkan flag unitbuf	Output
Uppercase	Mengaktifkan flag uppercase	Output
Ws	Tidak memperdulikan white-space	Input

Untuk menggunakan manipulator yang tercantum di atas kita harus memasukkan file *header* **iomanip.h**. (untuk C++ lama) atau **<iomanip>** (untuk C++ standar). Berikut ini contoh penggunaan salah satu manipulator di atas adalah sebuah program.

```
#include <iostream>
#include <iomanip>

using namespace std;
int main() {
  // Menggunakan flag setfill, setw dan endl
  cout<<setfill('*')<<setw(8)<<12<<endl;
  // Menggunakan flag oct dan endl
  cout<<oct<<64<<endl;
  // Menggunakan flag hex
  cout<<hex<<16<<endl;

  return 0;
}
```

Hasil yang akan diberikan dari program di atas adalah sebagai berikut:

```
******12
100
10
```

9.7 Input dan Output pada File

Dalam membuat program kadang kala kita sering menjumpai kasus yang berhubungan dengan proses input dan output file. Sebenarnya bentuk ini adalah bentuk penyederhanaan dari sistem I/O general, karena yang diolah hanyalah file. Di dalamnya telah didefinisikan fungsi-fungsi khusus untuk proses pemanipulasiannya. Maka dari itu, pada bagian ini kita kan membahas dasar-dasar yang terkandung di dalamnya.

9.7.1 Kelas yang Berhubungan dengan File

Seperti yang telah dikemukakan sebelumnya bahwa untuk menjalankan operasi I/ O kita jarus memasukkan file *header* **iostream.h** atau **<iostream>**. Terdapat beberapa buah kelas yang didefinisikan di dalamnya, yaitu **ifstream, ofstream,** dan **fstream.** Kelas-kelas ini merupakan turunan dari kelas **istream, ostream,** dan **iostream**. Seperti yang kita ketahui bahwa **istream, ostream,** dan **iostream** adalah kelas turunan dari kelas **ios**. Hal ini tentu akan menyebabkan kelas **ifstream**, **ofstream**, dan **fstream** juga dapat mengakses data-data dan semua operasi yang terdapat pada kelas ios.

9.7.2 Membuka dan Menutup File

Di dalam bahasa C++, kita membuka file dengan menghubungkannya ke sebuah stream. Ini berarti bahw sebelum kita dapat membuka file, kita harus mendapatkan stream terlebih dahulu. Seperti yang telah kita bhasa di atas bahwa terdapat tida macam stream, yaitu stream untuk proses input (ifstream), untuk proses input, maka harus kita deklarasikan dengan tipe ifstream. Begitu juga dengan variabel stream untuk proses output dan input/ output, masing-masing harus dideklarasikan sebagai ofstream dan fstream. Berikut ini contoh yang menunjukkan pendeklarasian varabel-variabel stream tersebut.

```
Ifstream input;          //variabel stream untuk proses input
Ifstream output          //variabel stream untuk proses output
Ifstream inOut;          //variabel stream untuk proses input/ output
```

Setelah mendeklarasikan variabel stream tersebut, maka langkah selanjutnya adalah menghubungkannya ke fie yaitu dengan cara memanggil fungsi **open().** Fungsi ini merupakan anggota dari masing-masing stream di atas. Berikut ini protoptype dari fungsi open() dalam masing-masing stream tersebut.

```
void  ifstream; open (const char* filename,
        ios:: openmode mode = ios:: in);
void ofstream:: open (const char* filename,
        ios::openmode mode = ios::out
ios : : trunc);
void fstream:: open (const char* filename,
        ios:: openmode mode = ios: : in | ios: :out);
```

Parameter *filename* di atas tidak lain adalah nama file yang akan dibuka termasuk lokasinya (path-nya). Sedangkan *mode* adalah menandakan bagaimana file tersebut akan dibuka. Seperti yang kita lihat pada prototype di atas bahwa kita dapat melakukan kombinasi dari nilai-nilai mode tersebut dengan melakukan operasi OR (menggunakan operator |) terhadapnya. Untuk lebih lengkapnya, berikut ini nilai-nilai dari mode di atas.

ios: : app, yang akan menyebabkan output dari file tersebut menjadi ditambahkan (*append*) pada bagian akhir baris. **ios : : ate,** akan menyebabkan pencarian ke akhir file ketika file tersebut dibuka.

ios : : in menandakan file tersebut mempunyai kapabilitas untuk input. Sedangkan **ios::out** adalah untuk output.

ios : : binary akan menyebabkan file yang akan dibuka tersebut dalam mode biner. Namun secara *default***,** file yang dibuka berada dalam mode teks.

Ios : : trunc akan menyebabkan isi file dengan nama yang sama dengan file yang telah dibuka, akan dipotong atau dibuang sehingga lebarnya menjadi nol.

Berikut ini contoh sintaks di dalam C++ yang berguna untuk membuka suatu file untuk proses output.

```
//Membuat atau mendeklarasikan variabel stream untuk proses output
ofstream output,

// memanggil fungsi open ()
Output.open (“myfile”, ios::out);
```

Kita dapat melakukan pencegahan jika ternyata file yang akan kita buka tidak ada atau terdapat kesalahan lainnya. Maka dari itu sebaiknya kita menambahkan suatu hadler pada sintaks di atas dengan menggunakan sintaks di bawah ini.

```
If (output) {
        cout <<”File tidak dapat dibuka”;
}
```

Setelah mengetahui cara untuk membuka file, maka di dalam program, kita tentu akan menutup file tersebut. Adapun cara yang digunakan untuk melakukan hal tersebut adalah dengan menggunakan fungsi **close()** . Tidak seperti fungsi **open (),** fungsi **close ()** ini tidak mempunyai parameter. Sebagai contoh, apabila kita telah menghubungkan file dengan stream yang bernama mystream, maka kita akan menutupnya dengan perintah berikut:

```
Mystream.close ();
```

9.7.3 Membaca dan Menuliskan Teks ke File

Membaca dan menuliskan teks ke file sama seperti kita melakukan input dan output ke terhadap I/O console, yaitu dengan menggunakan << dan >>. Perbedaannya di sini kita tidak menggunakan stream **cout** dan **cin**, melainkan menggunakan **stream** yang kita deklarasikan sendiri dan telah terhubung dengan sebuah file.

```
#include <iostream>
#include <fstream>

using namespace std;

int main() {
 // Mendeklarasikan stream untuk proses output
 ofstream output;
 output.open("D:/COBA.TXT");
 // Melakukan pencegahan terhadap terjadinya error
 if (!output) {
    cout<<"File tidak dapat dibuka"<<endl;
    return 1;
 }
```

lanjutan program…..

```
// Menuliskan teks ke dalam file tersebut
  output<<"Mengungkap Rahasia C++"<<endl;
  output<<"Oleh : Budi Raharjo"<<endl;
  output<<"Penerbit INFORMATIKA Bandung"<<endl;

  // Menutup file
  output.close();
  return 0;
}
```

Apabila program di atas dijalankan, maka kita akan melihat terbentuknya sebuah file baru bernama COBA.TXT di dalam drive D. Adapun isi file tersebut adalah sebagai berikut:

```
Mengungkap Rahasia C++
Oleh : Budi Raharjo
Penerbit INFORMATIKA Bandung
```

Contoh program melakukan proses input ke dalam sebuah variabel.

```
#include <iostream>
#include <fstream>
using namespace std;

int main() {
  // Mendeklarasikan stream untuk proses input
  ifstream input;
  // Membuka file yang dibuat pada program di atas
  input.open("D:/COBA.TXT");
  if (!input) {
     cout<<"File tidak dapat dibuka"<<endl;
     return 1;  }
      char S[100];
  input>>S;
  cout<<S<<endl;
  input>>S;
  cout<<S<<endl;
  input>>S;
  cout<<S<<endl;
  // Menutup file
  input.close();
  return 0;
}
```

Hasil yang akan diberikan dari program di atas adalah sebagai berikut

```
Mengungkap
Rahasia
C++
```

Hasil di atas menunjukkan bahwa kita telah melakukan input terhadap variabel S dengan nilai teks yang pertama kali di dalam file. Selanjutnya S diganti lagi nilainya dengan teks kedua di dalam file, begitu juga untuk nilai S yang ketiga. Hal inilah yang menyebabkan nilai S berubah-ubah pada saat ditampilkan ke layar.

9.8 Fungsi *put ()* dan *get ()*

Fungsi put () dan get () dapat juga digunakan untuk proses penulisan dan pembacaan data. Lebih tepatnya, untuk menuliskan data kita menggunakan fungsi put () dan utnuk membaca data kita menggunakan fungsi get(). Adapun bentuk umum dari penggunaan kedua buah fungsi ini adalah seperti yang terlihat di bawah ini.

```
ostream &put (char ch);
ostream &get (char& ch);
```

Berikut ini contoh program yang menunjukkan penggunaan fungsi put ().

```
#include <iostream>
#include <fstream>
using namespace std;

int main() {
  ofstream output;
  output.open("D:/TEST.TXT");
  if (!output) {
    cout<<"File tidak dapat dibuka"<<endl;
    return 1;  }
  int C=65;
  while  (char(C) <= 'Z') {
    output.put(char(C));
    C++;  }
  output.close();
  return 0;
}
```

Apabila program di atas dijalankan, maka akan terbentuk sebuah file baru yang bernama TEST.TXT di dalam *drive* D. Selanjutnya apabila kita buka file tersebut,

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

Setelah mengetahui penggunaan dari fungsi put (), maka selanjutnya kita akan menuliskan program yang akan menunjukkan penggunaan fungsi get(). Di sini, kita akan menggunakan file yang telah dibentuk sebelumnya, yaitu TEST.TXT. Adapun sintaks programnya dapat dilihat di bawah ini.

```
#include <iostream>
#include <fstream>

using namespace std;

int main() {

 ifstream input;

 input.open("D:/TEST.TXT");

 if (!input) {
  cout<<"File tidak dapat dibuka"<<endl;
  return 1;
 }

 char C;
 while  (input) {    // Selama belum EOF (End Of File)
  input.get(C);
  if (input)
    cout<<C;
 }

 input.close();

 return 0;
}
```

Hasil yang akan tampil di layar adalah sebagai berikut:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

Hasil di atas menunjukkan bahwa data yang tersimpan dalam file TEST.TXT (yang berbentuk karakter-karakter) satu per satu (melalui proses pengulangan) akan disimpan ke dalam variabel C dengan menggunakan fungsi get (). Sekanjutnya nilai C tersebut ditampilkan ke layar dengan menggunakan perintah cout.

9.9 Fungsi *write ()* dan *read ()*

Selain cara-cara yang telah disebutkan sebelumnya, terdapat cara lain untuk melakukan proses penulisan dan pembacaan data ke/ dari dalam file, yaitu dengan menggunakan fungsi write () dan read (). Fungsi-fungsi ini kebanyakan digunakan untuk file biner. Berikut ini bentuk umum dari penggunaan kedua buah fungsi tersebut.

```
ostream &write (const char *buf, streamsize);
istream &read (const char *buf, streamsize n);
```

Di sini fungsi write () akan menulis *n* buah karakter untuk dimasukkan ke stream dari *buffer* yang ditunjuk oleh pointer *buf*. Sedangkan fungsi read () akan membaca *n* buah karakter dari stream untuk ditempatkan ke *buffer* yang ditunjuk oleh pointer *buf*. Untuk lebih memahaminya, berikut ini contoh program yang menunjukkan penggunaan fungsi write () dan read ().

```
#include <iostream>
#include <fstream>
#include <cstring>

using namespace std;

struct SISWA {
  char NIM[9];
  char Nama[25];
  char Kota[15];
  int Usia;
};
```

lanjutan program….

```
// Mendeklarasikan stream untuk proses output
 ofstream OUTPUT;
 OUTPUT.open("D:/DATA", ios::out | ios::trunc | ios::binary);

 if (!OUTPUT) {
   cout<<"File tidak dapat dibuka"<<endl;
   return 1;
 }

 // Menuliskan ke stream
 OUTPUT.write((char *) &S, sizeof(S));

 OUTPUT.close();

 // Mendeklarasikan stream untuk proses input
 ifstream INPUT;
 INPUT.open("D:/DATA", ios::in | ios::binary);

 if (!INPUT) {
   cout<<"File tidak dapat dibuka"<<endl;
   return 1;
 }

 // Membaca dari stream ke buffer
 INPUT.read((char *) &S, sizeof(S));

 // Menampilkan data
 cout<<S.NIM<<endl;
 cout<<S.Nama<<endl;
 cout<<S.Kota<<endl;
 cout<<S.Usia<<endl;

 INPUT.close();
 return 0;
}
```

Apabila program du atas dijalankan, maka Anda akan melihat hasil sebagai berikut:

```
D0G86549
Budi Raharjo
Bandung
25
```

Hasil di atas merupakan hasil dari pengambilan data yang terdapat pada file DATA, yaitu biner yang sebelumnya dibentuk di dalam drive D. Adapun apabila kita lihat isi dari file biner tersebut (dengan memakai program atau aplikasi tertentu), maka contoh hasil yang akan diberikan adalah seperti yang tampak di bawah ini.

DAFTAR PUSTAKA

1. Rasihan Ari Yuono. 2005. Pemrograman C++.Diktat Mata Kuliah.UNS
2. Robbi Rohim. 2005. Perkembangan C++. Modul Pemrograman C++. STMIK Amikom.
3. Budi Raharjo. 2009. Pemrograman C++. Penerbit INFORMATIKA: Bandung